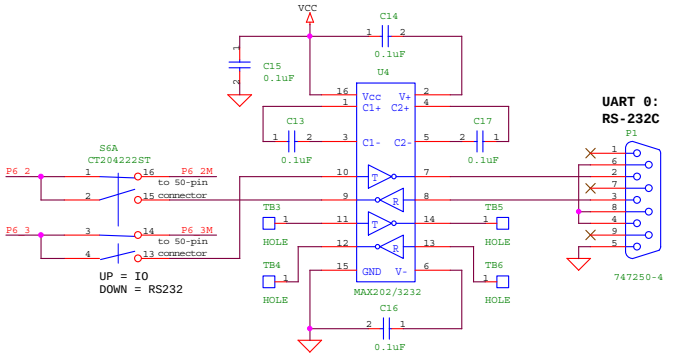
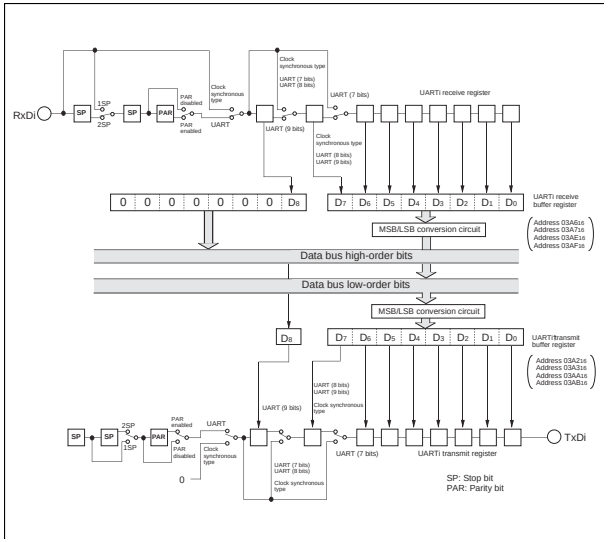
Polled Serial Communications

In these notes . . .

- Why serial?
- Serial Communications
  - RS232 standard
  - UART operation
  - Polled Code
  - Interrupt Driven Code
- Read M16C/62 Hardware Manual *(Serial I/O)*

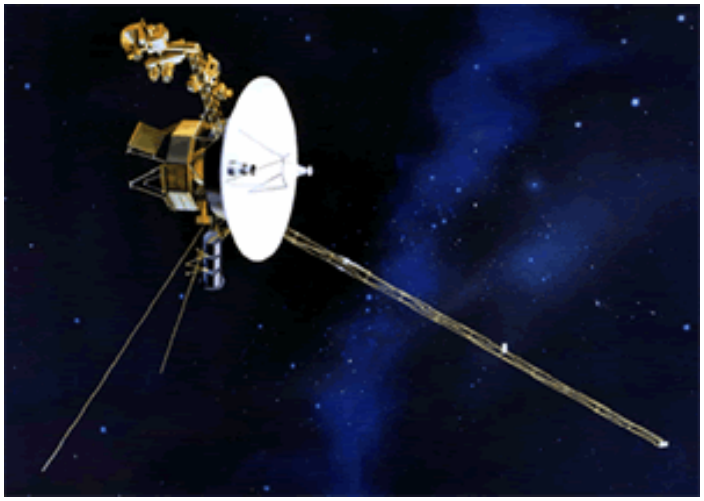




Why Communicate Serially?

- 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

Example System: Voyager Spacecraft



- Constraints: Reliability, power, size, weight, reliability, reliability, etc.
- "Uplink communications is via S-band (16-bits/sec command rate) while an X-band transmitter provides downlink telemetry at 160 bits/sec normally and 1.4 kbps for playback of high-rate plasma wave data. All data are transmitted from and received at the spacecraft via the 3.7 meter high-gain antenna (HGA)." http://voyager.jpl.nasa.gov/spacecraft/index.html
  - Uplink – to spacecraft
  - Downlink – from spacecraft

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

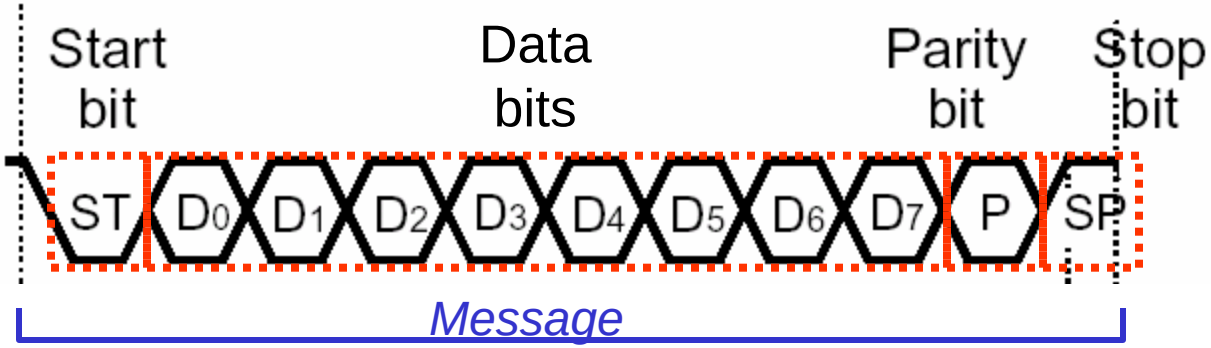
Asynchronous Serial Communication Basics

- 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.)

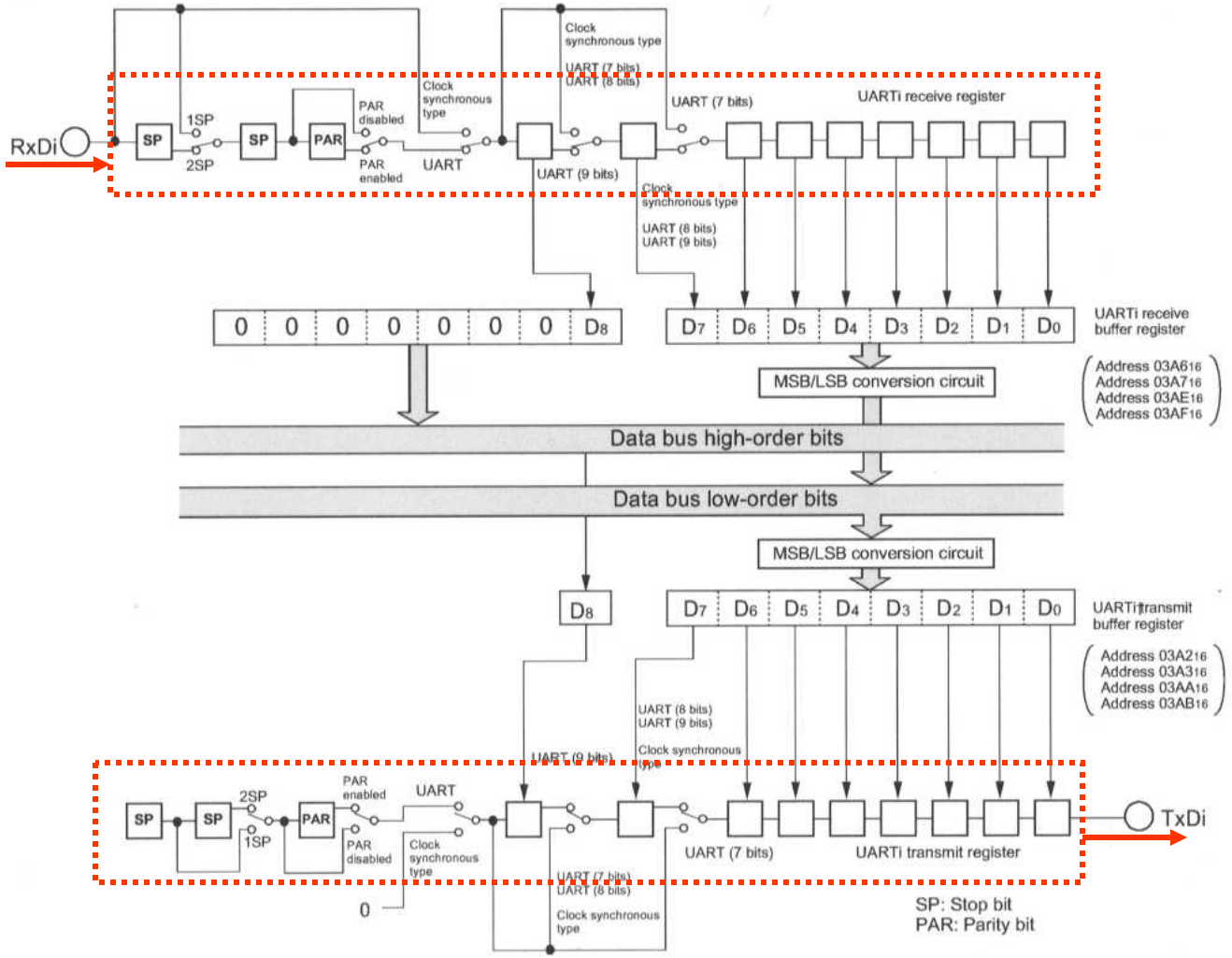
- More sophisticated network protocols have more information in each message
  - Medium access control – when multiple nodes are on bus, they must arbitrate for permission to transmit
  - Addressing information – for which node is this message intended?
  - Larger data payload
  - Stronger error detection 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 deserialize bits
  - Receiver/Transmitter
- M30626 has three
  - UART0, 1, and 2
  - UART1 talks to FoUSB-Mon circuit on back of board, enables us to do in-circuit debugging
  - Can operate in asynchronous or synchronous (not used here) modes
  - See MCU Hardware Manual for details, or else the remaining slides might be confusing

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



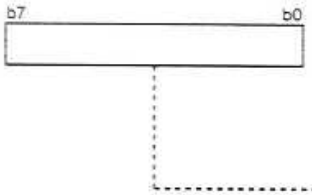
Setting up the Serial Port

- We will use UART 0, so all of the references to a UART will have “u0” in them.
- There are several control registers to set up before communicating
  - Set the port speed
  - Select 8 data bits, no parity, one stop bit (8N1)
  - Enable transmitter and receiver
- Code examples:
  - SerPoll, SerInt
  - Renesas application note and code for UART (check baud rate!)

Setting the Speed of the Serial Port

- Actual Baud rate = $F_{count\_source}/(16*(UiBRG+1))$
  - Multiple count sources available, based on system clock (20 MHz): f_1 (system clock), f_{8SIO} (1/8 system clock) and f_{32SIO} (1/32 system clock)
  - Note that ***CPU runs at 24 MHz***! Frequency doubler (PLL) derives a 24 MHz signal from the 12 MHz crystal
  - Solve for UiBRG = floor($F_{count\_source}/(16*(desired\ Baud\ Rate))+0.5$) - 1
- Desired Baud rate = 19,200 baud
  - 24 MHz/(16*19,200 baud) = 78.125
  - Load u0brg with 78 – 1 = 77
  - Actual baud rate = 24 MHz/(16*78) = 19230.77baud
    = 0.16% error
  - If error is too large, communication fails
  - This uses count source f1.

UARTi bit rate generator (Note 1, 2)



Symbol	Address	When reset
U0BRG	$03A1_{16}$	Indeterminate
U1BRG	$03A9_{16}$	Indeterminate
U2BRG	0379_{16}	Indeterminate

Function	Values that can be set	R	W
Assuming that set value = n, BRGi divides the count source by n + 1	00_{16} to FF_{16}	X	O

Note 1: Write a value to this register while transmit/receive halts.
Note 2: Use MOV instruction to write to this register.

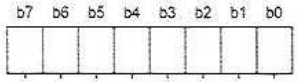
More Baud Rate Examples

- 24 MHz clock, 57600 baud

- 10 MHz clock, 1234 baud

UART0 Control Register 0

UARTi transmit/receive control register 0



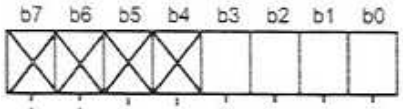
Symbol: UiC0(i=0,1) | Address: 03A4$_{16}$, 03AC$_{16}$ | When reset: 08$_{16}$

Bit symbol	Bit name	Function (During clock synchronous serial I/O mode)	Function (During UART mode)	R	W
CLK0	BRG count source select bit	b1 b0 0 0 : f1 is selected 0 1 : f8 is selected 1 0 : f32 is selected 1 1 : Inhibited	b1 b0 0 0 : f1 is selected 0 1 : f8 is selected 1 0 : f32 is selected 1 1 : Inhibited	○	○
CLK1				○	○
CRS	CTS/RTS function select bit	Valid when bit 4 = "0" 0 : CTS function is selected (Note 1) 1 : RTS function is selected (Note 2)	Valid when bit 4 = "0" 0 : CTS function is selected (Note 1) 1 : RTS function is selected (Note 2)	○	○
TXEPT	Transmit register empty flag	0 : Data present in transmit register (during transmission) 1 : No data present in transmit register (transmission completed)	0 : Data present in transmit register (during transmission) 1 : No data present in transmit register (transmission completed)	○	×
CRD	CTS/RTS disable bit	0 : CTS/RTS function enabled 1 : CTS/RTS function disabled (P60 and P64 function as programmable I/O port)	0 : CTS/RTS function enabled 1 : CTS/RTS function disabled (P60 and P64 function as programmable I/O port)	○	○
NCH	Data output select bit	0 : TXDi pin is CMOS output 1 : TXDi pin is N-channel open-drain output	0: TXDi pin is CMOS output 1: TXDi pin is N-channel open-drain output	○	○
CKPOL	CLK polarity select bit	0 : Transmit data is output at falling edge of transfer clock and receive data is input at rising edge 1 : Transmit data is output at rising edge of transfer clock and receive data is input at falling edge	Must always be "0"	○	○
UFORM	Transfer format select bit	0 : LSB first 1 : MSB first	Must always be "0"	○	○

Note 1: Set the corresponding port direction register to "0".
Note 2: The settings of the corresponding port register and port direction register are invalid.

UART0 Control Register 1

UARTi transmit/receive control register 1 (i=0, 1)

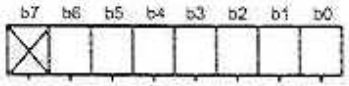


Symbol	Address	When reset
UiC1(i=0,1)	03A5₁₆,03AD₁₆	02₁₆

Bit symbol	Bit name	Function (During clock synchronous serial I/O mode)	Function (During UART mode)	R	W
TE	Transmit enable bit	0 : Transmission disabled 1 : Transmission enabled	0 : Transmission disabled 1 : Transmission enabled	○	○
TI	Transmit buffer empty flag	0 : Data present in transmit buffer register 1 : No data present in transmit buffer register	0 : Data present in transmit buffer register 1 : No data present in transmit buffer register	○	×
RE	Receive enable bit	0 : Reception disabled 1 : Reception enabled	0 : Reception disabled 1 : Reception enabled	○	○
RI	Receive complete flag	0 : No data present in receive buffer register 1 : Data present in receive buffer register	0 : No data present in receive buffer register 1 : Data present in receive buffer register	○	×
Nothing is assigned. In an attempt to write to these bits, write 0 . The value, if read, turns out to be 0 .				–	–

UART Control Register 2

UART transmit/receive control register 2



Symbol	Address	When reset
UCON	03B0₁₆	X0000000₂

Bit symbol	Bit name	Function (During clock synchronous serial I/O mode)	Function (During UART mode)	R	W
U0IRS	UART0 transmit interrupt cause select bit	0 : Transmit buffer empty (TI = 1) 1 : Transmission completed (TXEPT = 1)	0 : Transmit buffer empty (TI = 1) 1 : Transmission completed (TXEPT = 1)	○	○
U1IRS	UART1 transmit interrupt cause select bit	0 : Transmit buffer empty (TI = 1) 1 : Transmission completed (TXEPT = 1)	0 : Transmit buffer empty (TI = 1) 1 : Transmission completed (TXEPT = 1)	○	○
U0RRM	UART0 continuous receive mode enable bit	0 : Continuous receive mode disabled 1 : Continuous receive mode enable	Must always be 0	○	○
U1RRM	UART1 continuous receive mode enable bit	0 : Continuous receive mode disabled 1 : Continuous receive mode enabled	Must always be 0	○	○
CLKMD0	CLK/CLKS select bit 0	Valid when bit 5 = 1 0 : Clock output to CLK1 1 : Clock output to CLKS1	Invalid	○	○
CLKMD1	CLK/CLKS select bit 1 (Note)	0 : Normal mode (CLK output is CLK1 only) 1 : Transfer clock output from multiple pins function selected	Must always be 0	○	○
RCSP	Separate $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ bit	0 : $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ shared pin 1 : $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ separated	0 : $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ shared pin 1 : $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ separated	○	○
Nothing is assigned. In an attempt to write to this bit, write 0 . The value, if read, turns out to be indeterminate.				—	—

Note: When using multiple pins to output the transfer clock, the following requirements must be met:
* UART1 internal/external clock select bit (bit 3 at address 03A8₁₆) = 0 .

UART0 Tx/Rx Mode Register

UARTi transmit/receive mode register

b7	b6	b5	b4	b3	b2	b1	b0
0							

Symbol: UiMR(i=0,1) — Address: 03A0₁₆, 03A8₁₆ — When reset: 00₁₆

Bit symbol	Bit name	Function (During clock synchronous serial I/O mode)	Function (During UART mode)	R	W
SMD0	Serial I/O mode select bit	Must be fixed to 001 b2 b1 b0 0 0 0 : Serial I/O invalid 0 1 0 : Inhibited 0 1 1 : Inhibited 1 1 1 : Inhibited	b2 b1 b0 1 0 0 : Transfer data 7 bits long 1 0 1 : Transfer data 8 bits long 1 1 0 : Transfer data 9 bits long 0 0 0 : Serial I/O invalid 0 1 0 : Inhibited 0 1 1 : Inhibited 1 1 1 : Inhibited	○	○
SMD1				○	○
SMD2				○	○
CKDIR	Internal/external clock select bit	0 : Internal clock 1 : External clock	0 : Internal clock 1 : External clock (Note)	○	○
STPS	Stop bit length select bit	Invalid	0 : One stop bit 1 : Two stop bits	○	○
PRY	Odd/even parity select bit	Invalid	Valid when bit 6 = "1" 0 : Odd parity 1 : Even parity	○	○
PRYE	Parity enable bit	Invalid	0 : Parity disabled 1 : Parity enabled	○	○
Reserved		Must always be "0"		○	○

Note: Set the corresponding port direction register to 0.

Configuring UART0

```
void init_UART0() {
  // UART 0 baud rate gen.
  u0brg = (unsigned char) (
   f1_CLK_SPEED/(16*19200) - 1);

  // UART 0 tx/rx mode register
  smd2_u0mr = 1;   // 8 data bits
  smd1_u0mr = 0;
  smd0_u0mr = 1;
  ckdir_u0mr = 0; // intern. clk
  stps_u0mr = 0;
  pry_u0mr = 0;
  prye_u0mr = 0; // no parity

  // uart0 t/r control reg. 0
  // 20 MHz -> 19,200 baud
  clk1_u0c0 = 0; // sel. f/1 clk
  clk0_u0c0 = 0;
  nch_u0c0 = 0;
  // CMOS push-pull output
  ckpol_u0c0 = 0; // required
  uform_u0c0 = 0; // required
  crs_u0c0 = 0; // required
  crd_u0c0 = 1; // required

 // uart0 t/r control reg. 1
  te_u0c1 = 1; // enable tx
  re_u0c1 = 1; // enable rx

  // uart t/r control reg. 2
  u0irs = 0;   // select
   interrupt source
  u1rrm = 0;   // select
   interrupt source
  clkmd0 = 0;  // n/a
  clkmd1 = 0;  // n/a
  rcsp=1;     // rxdo port to
   p6_2
}
```

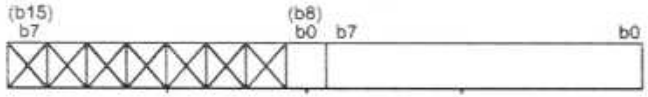
Using the UART

- When can we transmit?
  - Transmit buffer must be empty
  - Can poll `ti_u0c1` (UART0, control register 1, transmit buffer empty, 0x03A5, bit 1)
  - Or we can use an interrupt, in which case we will need to queue up data
- Put data to be sent into `u0tbl` (UART0, transmitter buffer, low byte, 0x03A2)
- Notice the differences between ones (`1`) and ells (`l`)

- When can we receive a byte?
  - Receive buffer must be full
  - Can poll `ri_u0c1` (UART0, control register 1, receive complete flag, 0x03A5, bit 3)
  - Or we can use an interrupt, and again we will need to queue the data
- Get data from `u0rbl` (UART0, receive buffer, low byte, 0x03A6)

Transmit Buffer

UARTi transmit buffer register (Note)



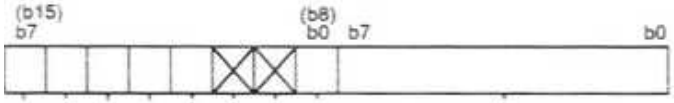
Symbol	Address	When reset
U0TB	$03A3_{16}$, $03A2_{16}$	Indeterminate
U1TB	$03AB_{16}$, $03AA_{16}$	Indeterminate
U2TB	$037B_{16}$, $037A_{16}$	Indeterminate

Function	R	W
Transmit data	X	O
Nothing is assigned. In an attempt to write to these bits, write 0 . The value, if read, turns out to be indeterminate.	—	—

Note: Use MOV instruction to write to this register.

Receive Buffer

UARTi receive buffer register



Symbol	Address	When reset
U0RB	$03A7_{16}$, $03A6_{16}$	Indeterminate
U1RB	$03AF_{16}$, $03AE_{16}$	Indeterminate
U2RB	$037F_{16}$, $037E_{16}$	Indeterminate

Bit symbol	Bit name	Function (During clock synchronous serial I/O mode)	Function (During UART mode)	R	W
—	———	Receive data	Receive data	○	×
Nothing is assigned. In an attempt to write to these bits, write 0 . The value, if read, turns out to be 0 .				–	–
ABT	Arbitration lost detecting flag (Note 2)	0 : Not detected 1 : Detected	Invalid	○	○
OER	Overrun error flag (Note 1)	0 : No overrun error 1 : Overrun error found	0 : No overrun error 1 : Overrun error found	○	×
FER	Framing error flag (Note 1)	Invalid	0 : No framing error 1 : Framing error found	○	×
PER	Parity error flag (Note 1)	Invalid	0 : No parity error 1 : Parity error found	○	×
SUM	Error sum flag (Note 1)	Invalid	0 : No error 1 : Error found	○	×

Note 1: Bits 15 through 12 are set to 0 when the serial I/O mode select bit (bits 2 to 0 at addresses $03A0_{16}$, $03A8_{16}$ and 0378_{16}) are set to 000_2 or the receive enable bit is set to 0 .
(Bit 15 is set to 0 when bits 14 to 12 all are set to 0 .) Bits 14 and 13 are also set to 0 when the lower byte of the UARTi receive buffer register (addresses $03A6_{16}$, $03AE_{16}$ and $037E_{16}$) is read out.

Note 2: Arbitration lost detecting flag is allocated to U2RB and only "0" must be written. Nothing is assigned in bit 11 of U0RB and U1RB. Set to "0" when writing. If read, value is "0".

Example 1: Send Out Many Characters

- Send out every character and symbol from ‘A’ to ‘z’ and then repeat
- Use polling to determine when transmit buffer is empty and can be reloaded

```
void demo1() {
  // polled transmit demo
  unsigned long ctr;
  char c='A';
  while (1) {
    while (!ti_u0c1); // wait for
        // transmit buffer empty
    for (ctr=0; ctr<1000; ctr++);
        // delay so the letters
        // come out slowly
    u0tbl = c; // load c into
        // transmit buffer
    c++; // !
    if (c>'z')
      c = 'A'; // wrap around
  }
}
```

Example 2: Make the Code More Elegant

- Create a function to transmit a null-terminated string

```
void demo2_ser_tx(far char * buf)
   {
  // polled buffer transmit demo
  // transmit a null-terminated
  // string
  while (*buf) {
    while (!ti_u0c1); // wait for
        // transmitter empty
    u0tbl = *buf; // load
        // character into buffer
    buf++; // increment buffer
        // pointer
  }
}
void demo2() {
  while (1) {
    demo2_ser_tx("Testing!\n\r");
    delay(100000);
  }
}
```

Example 3: Echo Received Data

- Wait for character to be received (via polling)
- Add one to character if switch 2 is pressed (LSSC (laughably simple substitution cipher))
- Wait for transmitter to be empty (ready)
- Send character
- If the character is a “return” send a line feed to make terminal scroll to new line

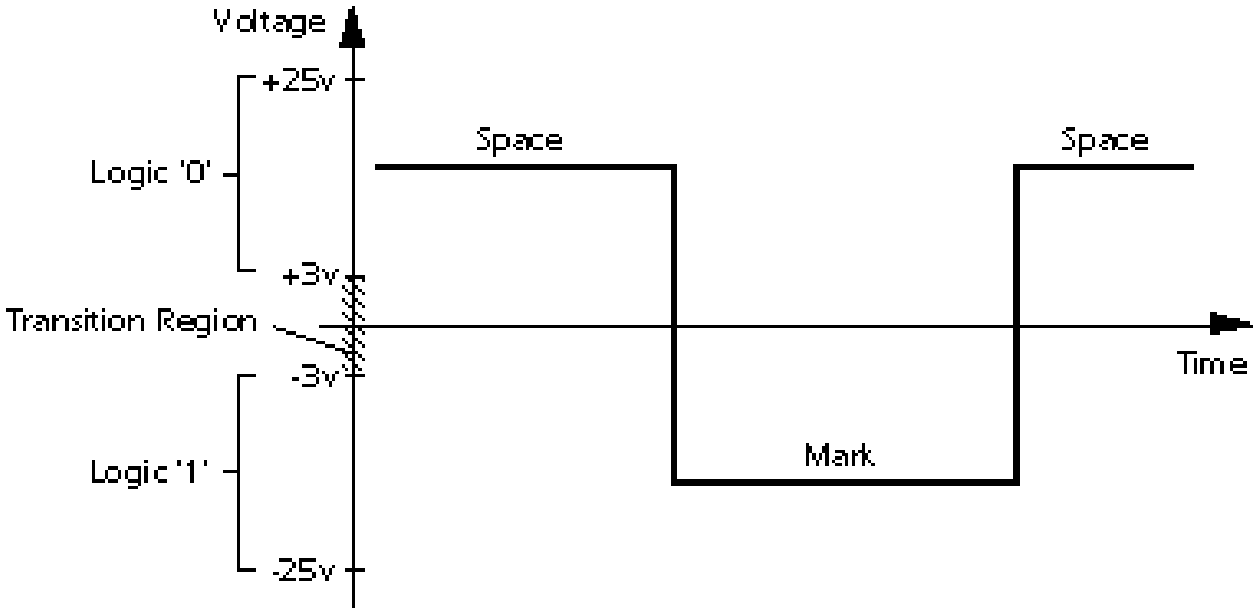
```
void demo3() {
  //  polling echo example
  char c;
  while (1) {
    while (!ri_u0c1) // await rx
      ;
    c = u0rbl; // get character
    if (!S2) // "Encrypt“ c
      c += 1;
    while (!ti_u0c1) // await tx
      ;
    u0tbl = c; // send character
    // cr/lf translation for terminal
    if (c == '\r') {
      while (!ti_u0c1)
        ;
      u0tbl = '\n';
    }
  }
}
```

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 may be different
  - Extra symbols (channel changes) may be inserted for framing, error detection, acknowledgment, etc. These ***reduce*** the bit rate
  - A single symbol might encode more than one bit. This ***increases*** the bit rate.
    - E.g. multilevel signaling, quadrature amplitude modulation, phase amplitude modulation, etc.

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 QSK62P