Interrupt-Driven Serial Communication

Lecture 12

In these notes . . .

Interrupt design guidelines

Interrupt-driven serial I/O device driver design

- Interrupt map for system
- Enabling interrupts
- Queue concepts and implementation

Interrupt Design Guidelines

(Ganssle, p. 57)

Create and Maintain an Interrupt Map

- Make a spreadsheet showing interrupts for system, maximum rate, maximum latency allowed. Fill in with execution time measurements during development.
- This provides a CPU time budget to follow, and lets us predict worst-case values:
  - CPU utilization from interrupts
  - Time interrupts are disabled

Keep ISRs short. Leave lengthy work to task code when possible. E.g. converting time ticks to H:M:S

Keep ISRs so trivially simple that bugs are rare.

Fill all unused interrupt vectors with a pointer to a debug routine which hangs and flashes a light. This way you'll find out about unplanned interrupts immediately.

Example Interrupt Map

Interrupt	Maximum Latency	Maximum Frequency	Maximum Duration	Activity Description
UART 0 Receive	one character time = 1/1920 = 520.8 μs	19200 Hz/10 = 1920 Hz	*fill in as we develop code*	Enqueue received character
UART 0 Transmit	None, but performance degrades	19200 Hz/10 = 1920 Hz	*fill in as we develop code*	Dequeue and send outgoing character
Timer Overflow	1 clock tick = 41 ns	24 MHz/65536 = 366.2 Hz	*fill in as we develop code*	Increment timer extension tchi
UART 1 Receive	one character time	baud rate/10		
UART 1 Transmit	None, but performance degrades	baud rate/10		

Interrupt Performance Examination

Measurement

- Use scope to measure duration of tx_isr, rx_isr by examining output bits
- Set scope to “infinite persistence” to capture all events
- Could also use get_ticks to automate data capture

Analysis

- How long do the ISRs last?
- Is there any variation? If so, why?
- How long is the delay between receiving a character and sending out the reply?

Performance Prediction

- Max. CPU loading from interrupt = max. interrupt frequency * worst-case ISR duration
- Min. period between interrupts = 1/max. frequency (if periodic, otherwise need to find minimum interarrival period)
- Add up all ISRs which could be requested simultaneously (*critical instant*). These delay system response.
- Overrun risk: CPU doesn’t finish current ISR before another interrupt *of the same type* occurs
- Deadline risk: CPU doesn’t start an ISR before reaching its deadline.

If we use a spreadsheet…

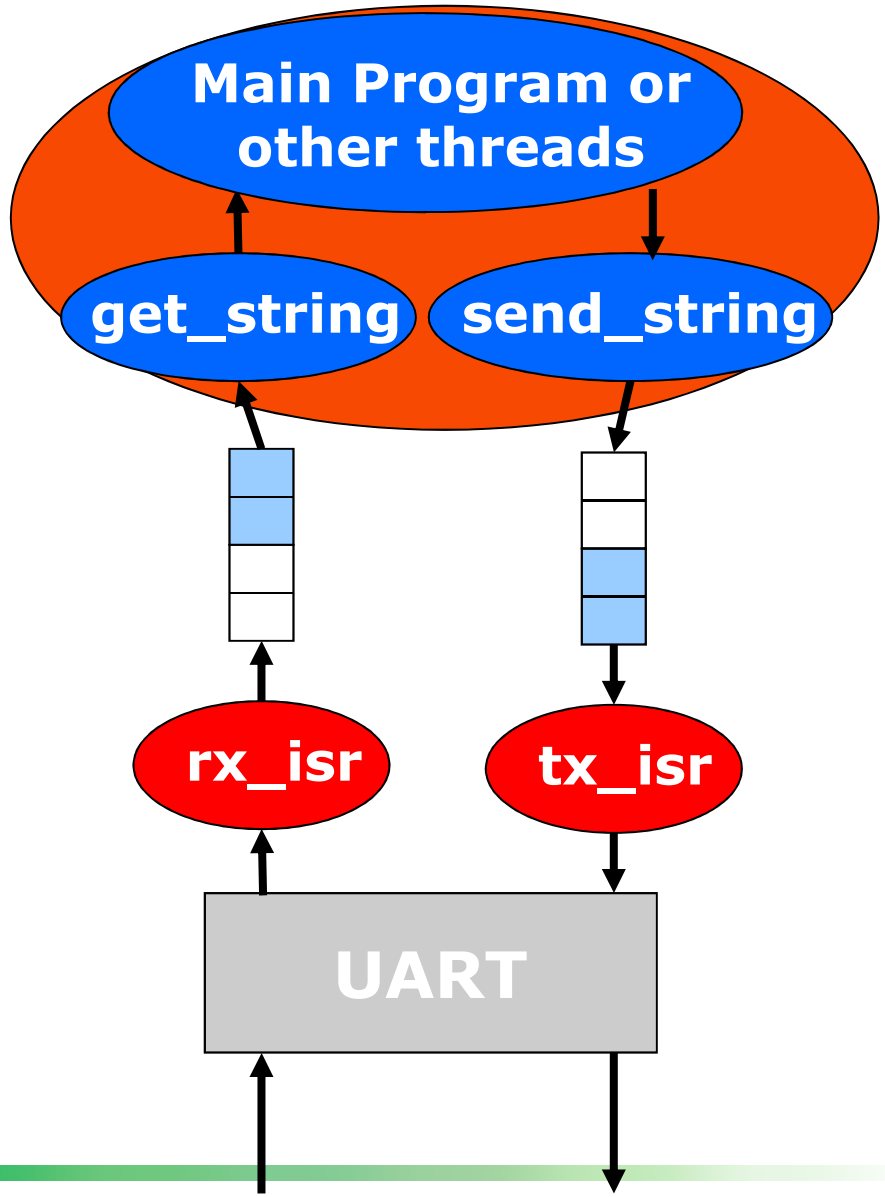
Serial Communications and Interrupts

Now we have three separate threads of control in the program

- main program (and subroutines it calls)
- Transmit ISR – executes when UART is ready to send another character
- Receive ISR – executes when UART receives a character

Need a way of buffering information between threads

- Solution: circular queue with head and tail pointers
- One for tx, one for rx



Enabling and Connecting Interrupts to ISRs

Need to enable UART's interrupts: send and receive

Page 49: lists many interrupt sources, but nothing says UART!

Investigate sfr62p.h instead, search for UART

- sxtic and sxric (x = 0, 1, 2) are the interrupt registers for serial communications (UART 0, 1 and 2)

Set their priorities to >0 to enable them

Create shells for ISRs, fill in later

Don't forget to modify interrupt vectors 17 and 18 in sect30.inc to point to u0_tx_isr and u0_rx_isr

```
init_UART0() {
        …
        // code deleted for clarity
        s0ric = 5; // enable UART 0
        // receive interrupt with
        // priority 5

        s0tic = 4; // enable UART 0
        // transmit interrupt with
        // priority 4
        …
}
#pragma INTERRUPT u0_tx_isr
void u0_tx_isr() {
        // add code here!
}
#pragma INTERRUPT u0_rx_isr
void u0_rx_isr() {
        // add code here!
}
```

Code to Implement Queues

Enqueue at tail (tail_ptr points to next free entry), dequeue from head (head_ptr points to item to remove)

#define the queue size to make it easy to change

One queue per direction

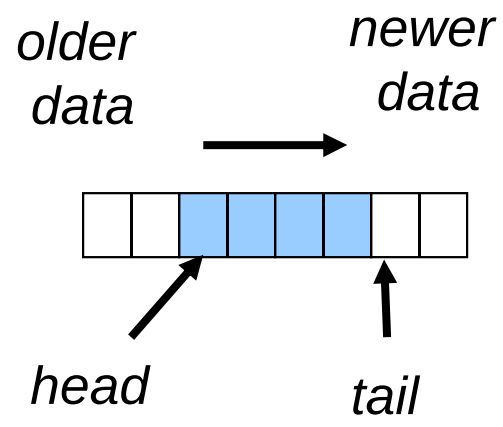
- tx ISR unloads tx_q
- rx ISR loads rx_q

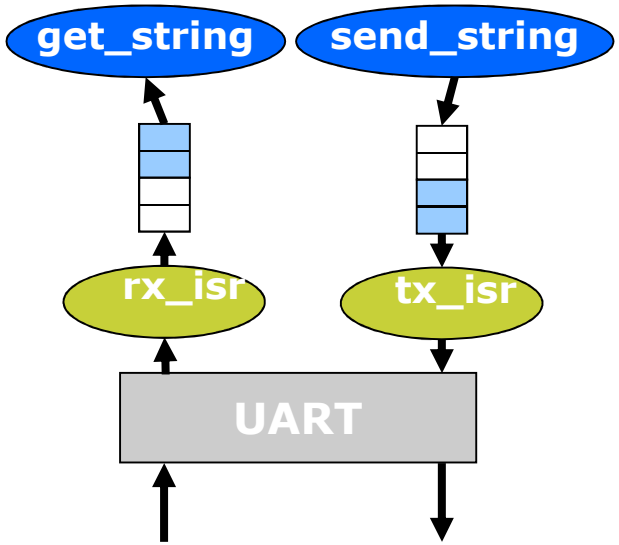
Other threads (e.g. main) load tx_q and unload rx_q

Need to wrap pointer at end of buffer to make it circular, use % (modulus, remainder) operator

Queue is empty if size == 0

Queue is full if size == Q_SIZE





Defining the Queues

```
#define Q_SIZE (32)

typedef struct {
  unsigned char Data[Q_SIZE];
  unsigned int Head; // points to oldest data element
  unsigned int Tail; // points to next free space
  unsigned int Size; // quantity of elements in queue
} Q_T;

Q_T tx_q, rx_q;
```

Initialization and Status Inquiries

```
void Q_Init(Q_T * q) {
  unsigned int i;
  for (i=0; i<Q_SIZE; i++)
    q->Data[i] = 0;  // to simplify our lives when debugging
  q->Head = 0;
  q->Tail = 0;
  q->Size = 0;
}

int Q_Empty(Q_T * q) {
  return q->Size == 0;
}

int Q_Full(Q_T * q) {
  return q->Size == Q_SIZE;
}
```

Enqueue and Dequeue

```
int Q_Enqueue(Q_T * q, unsigned char d) {
  // What if queue is full?
  if (!Q_Full(q)) {
    q->Data[q->Tail++] = d;
    q->Tail %= Q_SIZE;
    q->Size++;
    return 1; // success
  } else
    return 0; // failure
}
unsigned char Q_Dequeue(Q_T * q) {
  // Must check to see if queue is empty before dequeueing
  unsigned char t=0;
  if (!Q_Empty(q)) {
    t = q->Data[q->Head];
    q->Data[q->Head++] = 0; // to simplify debugging
    q->Head %= Q_SIZE;
    q->Size--;
  }
  return t;
}
```

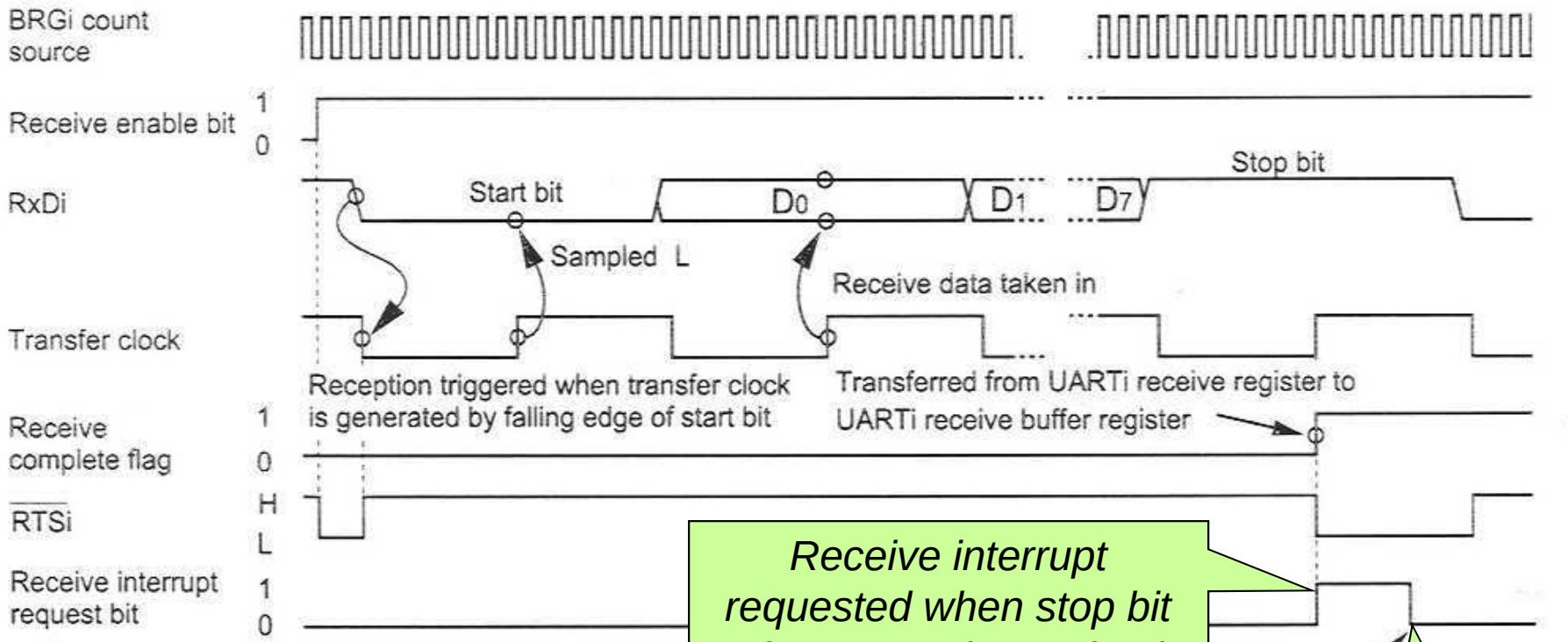
Now the ISRs

```
#pragma INTERRUPT u0_tx_isr
void u0_tx_isr() {
  LED_G = LED_ON;
  if (!Q_Empty(&tx_q))
    u0tbl = Q_Dequeue(&tx_q);
  LED_G = LED_OFF;
}

#pragma INTERRUPT u0_rx_isr
void u0_rx_isr() {
  LED_Y = LED_ON;
  if (!Q_Enqueue(&rx_q, u0rbl)) {
    LED_R = LED_ON;
  }
  LED_Y = LED_OFF;
}
```

Receive Interrupt Example

• Example of receive timing when transfer data is 8 bits long (parity disabled, one stop bit)

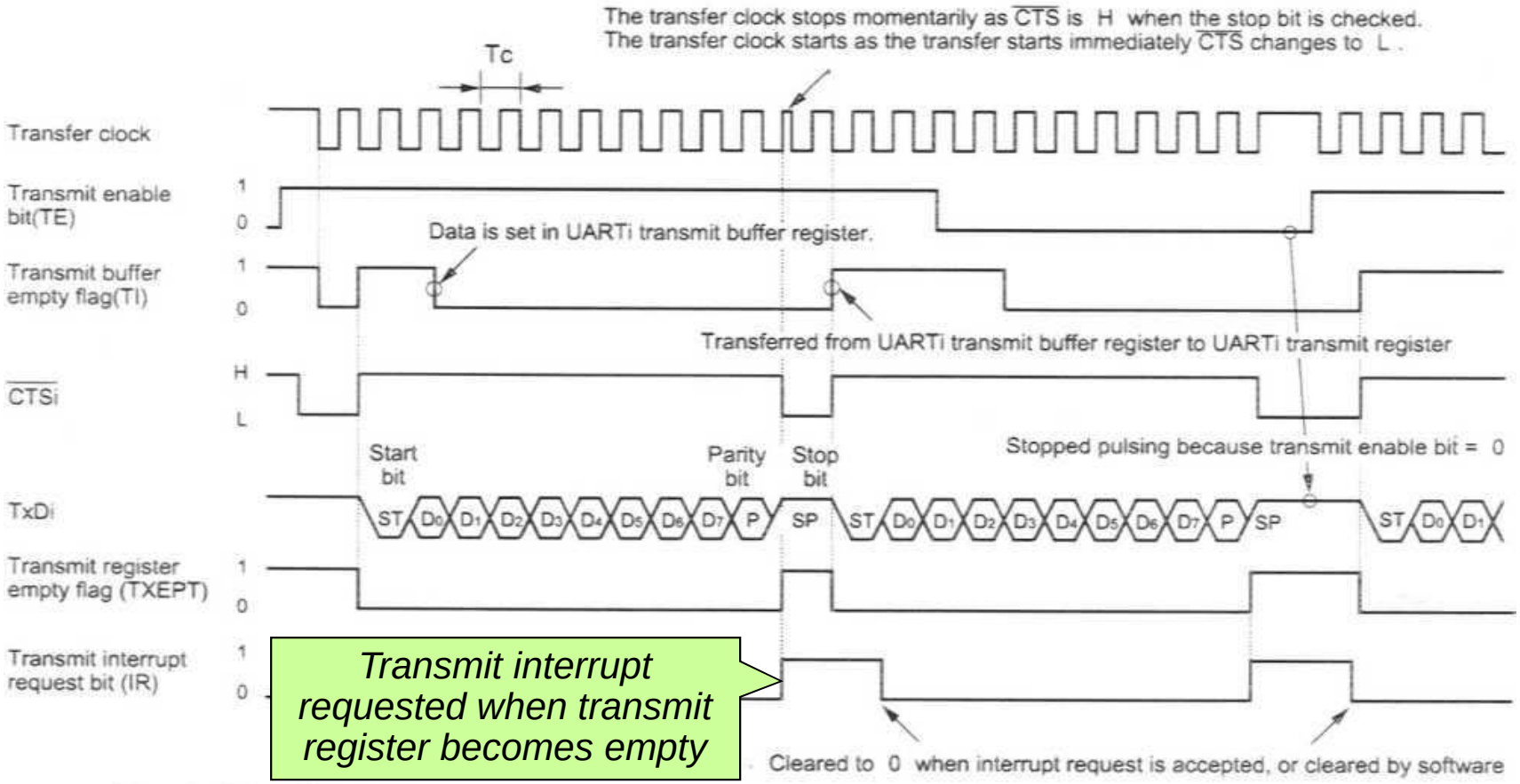


The above timing applies to the following settings :

* Parity is disabled.
* One stop bit.
* $\overline{\text{RTS}}$ function is selected.

Transmit Interrupt Example

- Example of transmit timing when transfer data is 8 bits long (parity enabled, one stop bit)



Shown in ( ) are bit symbols.

The above timing applies to the following settings :
- Parity is enabled.
- One stop bit.
- CTS function is selected.
- Transmit interrupt cause select bit = 1 .

Tc = 16 (n + 1) / fi or 16 (n + 1) / fEXT
fi : frequency of BRGi count source (f1, f8, f32)
fEXT : frequency of BRGi count source (external clock)
n : value set to BRGi

Tying Things Together

```
void d3_send_string(far char * s) {
  // enqueue a null-terminated string
  // for serial tx
  while (*s) {
    if (Q_Enqueue(&tx_q, *s))
      s++;
    else {
       // queue is full,
              // wait for some time
              // or signal an error?
    }
  }
  // if transmitter not busy,
  // get it started
  if (ti_u0c1) {
    u0tbl = Q_Dequeue(&tx_q);
  }
}
```

Create a function to load up the transmit queue with a string

Issue: What should the system do if the transmit queue is full?

We also need to start UART transmitting if it isn't already

Another Example

You can use this code as another example.

```
//  Xmit_intr -Interrupt handler for the transmit buffer.
void   Xmit_intr(void) {
  if(SXMIT_OUT != SXMIT_IN){ // more characters to transmit.
    _u0tb = Ser_XMIT_q[SXMIT_OUT];
    if(++SXMIT_OUT >= XMIT_BUFFER_SIZE)
      SXMIT_OUT = 0;
    }
  else {              // transmit buffer empty
    _u0c1 &= 0x04;  // transfer disable
    asm("FCLR I");  // disable interrupts
    _s0tic = 0x00;  // disable transmit interrupt
    asm("FSET I");  // enable interrupts
  }
}
```