Last Time

Bit manipulation
• Determining pin configuration: input/output
• Determining the output pin state
• Reading the input pin
Today

• Communicating between devices
  – Serial communication
  – Communication in code

• Project 1 is due on Tuesday: don’t delay on getting started

• Next Thursday’s class: in the lab
Input/Output Systems

Processor needs to communicate with other devices:

- Receive signals from sensors
- Send commands to actuators
- Or both (e.g., disks, audio, video devices)
I/O Systems

Communication can happen in a variety of ways:

- Binary parallel signal (e.g., project 1)
- Analog
- Serial signals
An Example: SICK Laser Range Finder

• Laser is scanned horizontally
• Using phase information, can infer the distance to the nearest obstacle
• Resolution: ~.5 degrees, 1 cm
• Can handle full 180 degrees at 20 Hz
Serial Communication

• Communicate a set of bytes using a single signal line

• We do this by sending one bit at a time:
  – The value of the first bit determines the state of a signal line for a specified period of time
  – Then, the value of the 2\textsuperscript{nd} bit is used
  – Etc.
Serial Communication

The sender and receiver must have some way of agreeing on when a specific bit is being sent

- Typically, each side has a clock to tell it when to write/read a bit
- In some cases, the sender will also send a clock signal (on a separate line)
- In other cases, the sender/receiver will first synchronize their clocks before transfer begins
Asynchronous Serial Communication

• The sender and receiver have their own clocks, which they do not share
• This reduces the number of signal lines
• Bidirectional transmission, but the two halves do not need to be synchronized in time

But: we still need some way to agree that data is valid. How?
Asynchronous Serial Communication

How can the two sides agree that the data is valid?

• Must both be operating at essentially the same transmit/receive frequency

• A data byte is prefaced with a bit of information that tells the receiver that data is coming

• The receiver uses the arrival time of this start bit to synchronize its clock
A Typical Data Frame

The stop bits allow the receiver to immediately check whether this is a valid frame

- If not, the byte is thrown away
Data Frame Handling

Most of the time, we do not personally deal with the data frame level. Instead, we rely on:

• Hardware solutions: Universal Asynchronous Receiver Transmitter (UART)
  – Very common in computing devices
• Software solutions in libraries
One Standard: RS232-C

Defines a logic encoding standard:

- “High” is encoded with a voltage of -5 to -15 (-12 to -13V is typical)
- “Low” is encoded with a voltage of 5 to 15 (12 to 13V is typical)
RS232-C

Originally intended to connect:

• Data Terminal Equipment (DTE)
  – Teletypes

• to Data Communication Equipment (DCE)
  – Modems

Now that we are connecting a computer to some peripheral, it is not always clear which is the DTE and which is the DCE.
RS232-C

Defines a pin assignment standard. For example, with the DB-9 connectors:

• Pin 2: receive (to DTE from DCE)
• Pin 3: transmit (from DTE to DCE)
• Pin 5: common (ground)

Also common to have DB-25 connectors (older standard)
RS232 on the Mega8

Our mega 8 has a Universal, Asynchronous serial Receiver/Transmitter (UART)

• Handles all of the bit-level manipulation
• You only have to interact with it on the byte level
Mega8 UART C Interface

OUlib support:
serial0_init(9600): initialize the port @9600 bits per second
getchar(): receive a character
kbhit(): is there a character in the buffer?
putchar(0x45): put a character out to the port

See the Atmel HOWTO
Character Representation

• A “char” is just an 8-bit number
• In some cases, we just interpret it differently.
• But: we can still perform mathematical operations on it
## Character Representation: ASCII

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>010 0000</td>
<td>32</td>
<td>20</td>
<td>$^\circ$</td>
<td>100 0000</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>110 0000</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>010 0001</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>100 0001</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>110 0001</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>010 0010</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>100 0010</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>110 0010</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>010 0011</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>100 0011</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>110 0011</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>010 0100</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>100 0100</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>110 0100</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>010 0101</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>100 0101</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>110 0101</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>010 0110</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>100 0110</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>110 0110</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>010 0111</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>100 0111</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>110 0111</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>010 1000</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>100 1000</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>110 1000</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>010 1001</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>100 1001</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>110 1001</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>010 1010</td>
<td>42</td>
<td>2A</td>
<td>^</td>
<td>100 1010</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>110 1010</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>010 1011</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>100 1011</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>110 1011</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>010 1100</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>100 1100</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>110 1100</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>010 1101</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>100 1101</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>110 1101</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>010 1110</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>100 1110</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110 1110</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>010 1111</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>100 1111</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>110 1111</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>011 0000</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>101 0000</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>111 0000</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>011 0001</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>101 0001</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>111 0001</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>011 0010</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>101 0010</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>111 0010</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>011 0011</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>101 0011</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>111 0011</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>011 0100</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>101 0100</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>111 0100</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>011 0101</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>101 0101</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>111 0101</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>011 0110</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>101 0110</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>111 0110</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>011 0111</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>101 0111</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>111 0111</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>011 1000</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>101 1000</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>111 1000</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>011 1001</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>101 1001</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>111 1001</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>011 1010</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>101 1010</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>111 1010</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>011 1011</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>101 1011</td>
<td>91</td>
<td>5B</td>
<td>]</td>
<td>111 1011</td>
<td>123</td>
<td>7B</td>
<td>]</td>
</tr>
<tr>
<td>011 1100</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>101 1100</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>111 1100</td>
<td>124</td>
<td>7C</td>
<td>\</td>
</tr>
<tr>
<td>011 1101</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>101 1101</td>
<td>93</td>
<td>5D</td>
<td></td>
<td>111 1101</td>
<td>125</td>
<td>7D</td>
<td></td>
</tr>
<tr>
<td>011 1110</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>101 1110</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>111 1110</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
</tbody>
</table>
Mega8 UART
Mega8 UART

- Transmit pin (PD1)
Mega8 UART

- Transmit pin (PD1)
- Transmit shift register
Writing a Byte to the Serial Port

```c
putchar('A');
```
Transmit

`putchar('A');`
Transmit

When UART is ready, the buffer contents are copied to the shift register.
Transmit

The least significant bit (LSB) of the shift register determines the state of the pin.

01000001

1
Transmit

After a delay, the UART shifts the values to the right

\[ x = \text{value doesn't matter} \]
Transmit

Next shift
Transmit

Several shifts later…
Receive

- Receive pin (PD0)
Receive

- Receive pin (PD0)
- Receive shift register
Receive

- “1” on the pin
- Shift register initially in an unknown state
Receive

“1” is presented to the shift register...
Receive

“1” is shifted into the **most significant bit (msb)** of the shift register
Receive

Next bit is shifted in
Receive

And the next bit...
Receive

And the 8\textsuperscript{th} bit
Receive

Completed byte is stored in the UART buffer
Reading a Byte from the Serial Port

```c
int c;

c=getchar();
```
Receive

`getchar()` retrieves this byte from the buffer
Reading a Byte from the Serial Port

```c
int c;

c=getchar();
```

Note: getchar() “blocks” until a byte is available

• Will only return with a value once one is available to be returned
Processing Serial Input

```c
int c;
while(1) {
    if(kbhit()) {
        // A character is available for reading
        c = getchar();
        //do something with the character>
    }
    //do something else while waiting>
}

kbhit() tells us whether a byte is ready to be read
```
Mega8 UART C Interface

`printf()`: formatted output

`scanf()`: formatted input

See the LibC documentation or the AVR C textbook

Note: `scanf()` does not work properly with `serial0_init()` (more on this later)
Serial I/O by Polling

```c
int c;
while(1) {
    if(kbhit()) {
        // A character is available for reading
        c = getchar();
        <do something with the character>
    }
    <do something else while waiting>
}
```
Next Time

• Building circuits with Atmel mega8s
• Getting ready for project 2
Last Time

- Interrupts in general
- External interrupt request
  - The mega8 has 2 pins
- Serial protocols
- RS232-C
Today

- Serial processing: from polling to interrupts
I/O By Polling

Polling works great … but:
I/O By Polling

Polling works great … but:
• We have to guarantee that our other tasks do not take too long (otherwise, we may miss the event)
• Depending on the device, “too long” may be very short
Serial I/O by Polling

```c
int c;
while(1) {
    if(kbhit()) {
        // A character is available for reading
        c = getchar();
        <do something with the character>
    }
    <do something else while waiting>
}
```

With this solution, how long can “something else” take?
I/O by Polling

In practice, we typically reserve this polling approach for situations in which:

• We know the event is coming very soon
• We must respond to the event very quickly

(both are measured in nano- to micro-seconds)
Receiving Serial Data

How can we allow the “something else” to take a longer period of time?
Receiving Serial Data

How can we allow the “something else” to take a longer period of time?

• The UART implements a 1-byte buffer
• Let’s create a larger buffer…
Receiving Serial Data

Creating a larger (circular) buffer. This will be a globally-defined data structure composed of:

- **N-byte memory space:**
  ```
  char buffer[BUF_SIZE];
  ```

- **Integers that indicate the first element in the buffer and the number of elements:**
  ```
  uint8_t front, nchars;
  ```
Buffered Serial Data

Implementation:

• We will use an interrupt routine to transfer characters from the UART to the buffer as they become available

• Then, our main() function can remove the characters from the buffer
Interrupt Handler

// Called when the UART receives a byte
ISR(USART_RXC_vect) {
    // Handle the character in the UART buffer
    if (nchars == BUF_SIZE) {
        getchar();
    } else {
        uint8_t i = (front + nchars) % BUF_SIZE;
        buffer[i] = getchar();
        ++nchars;
    }
}

Andrew H. Fagg: Embedded Real-Time Systems: Serial Comm
Interrupt Handler

// Called when the UART receives a byte
ISR(USART_RXC_vect) {
    // Handle the character in the UART buffer
    int c = getchar();

    if(nchars < BUF_SIZE) {
        buffer[(front+nchars)%BUF_SIZE] = c;
        nchars += 1;
    }
}
Reading Out Characters

// Called by a "main" program
// Get the next character from the circular buffer
int get_next_character() {
    if(nchars == 0)
        return(-1);  // No characters
    else{
        // Return the next character
        int tmp = buffer[front];
        front = (front + 1)%BUF_SIZE;
        --nchars;
        return(tmp);
    }
}
Last Time

- Interrupt Service Routines
- Circular buffers
  - Also known as “First In-First Out” queues
  - ISR filled the buffer as soon as serial data came in
  - Main program removed characters as needed
Today

- The shared data problem
  - Can occur when an ISR and the main program access and modify the same data structures
- Finite state machines
  - Expressing sequential behavior
Reading Out Characters

// Called by a “main” program
// Get the next character from the circular buffer
int get_next_character() {
    int c;
    if(nchars == 0)
        return(-1); // Error
    else {
        // Pull out the next character
        c = buffer[front];

        // Update the state of the buffer
        --nchars;
        front = (front + 1)%BUF_SIZE;
        return(c);
    }
}
int c;
while(1) {
    do {
        ????
    }while(???);
    <do something else while waiting>
}
An Updated main()

```c
int c;
while(1) {
    do {
        c = get_next_character();
        if(c != -1)
            <do something with the character>
    }while(c != -1);

    <do something else while waiting>
}
```
Buffered Serial Data

This implementation captures the essence of what we want, but there are some subtle things that we must handle ....
Buffered Serial Data

Subtle issues:

• The reading side of the code must make sure that it does not allow the buffer to overflow
  – But at least we have BUF_SIZE times more time before this happens

• We also have a shared data problem …
The Shared Data Problem

• Two independent segments of code that could access the same data structure at arbitrary times

• In our case, get_next_character() could be interrupted while it is manipulating the buffer
  – This can be very bad
Solving the Shared Data Problem

• There are segments of code that we want to execute without being interrupted

• We call these code segments critical sections
Solving the Shared Data Problem

There are a variety of techniques that are available:

- Clever coding
- Hardware: test-and-set instruction
- Semaphores: software layer above test-and-set
- Disabling interrupts
Solving the Shared Data Problem

There are a variety of techniques that are available:

• Clever coding
• Hardware: test-and-set instruction
• Semaphores: software layer above test-and-set
• Disabling interrupts
Disabling Interrupts

• How can we modify get_next_character()?

• It is important that the critical section be as short as possible

Assume:

• serial_receive_enable(): enable interrupt flag
• serial_receive_disable(): clear (disable) interrupt flag
Modified `get_next_character()`

```c
int get_next_character() {
    int c;
    serial_receive_disable();
    if(nchars == 0)
        serial_receive_enable();
    return(-1); // Error
    else {
        // Pull out the next character
        c = buffer[front];
        --nchars;
        front = (front + 1)%BUF_SIZE;
        serial_receive_enable();
        return(c);
    }
}
```
Initialization Details

```c
main()
{
    nchars = 0;
    front = 0;

    // Enable UART receive interrupt
    serial_receive_enable();

    // Enable global interrupts
    sei();
    :
```
Enabling/Disabling Interrupts

• Enabling/disabling interrupts allows us to ensure that a specific section of code (the critical section) cannot be interrupted
  – This allows for safe access to shared variables

• But: must not disable interrupts for a very long time
Final Note

For what we are doing:

• ISRs are not interrupted
• This means that the ISR is already a critical section