Microcontrollers

Class 3: Interrupts

March 21, 2011

Outline

Outline

Outline

Interrupts

What are they?

- Functions that get called (automatically) when a certain condition is true (Interrupt Service Routines)
- Pin-change (external) interrupts, data-ready interrupts, timer-driven interrupts
- See p. 59 of the datasheet, and read `iomx8.h` for their names

What are they good for?

- Respond to external stimulus in real-time: user pressed button? new serial data came in? robot sensed a wall ahead?
- Clean up our code: event-driven programming

Polling

Life before Interrupts

- Wanted to see if a button was pressed, we checked the button state every time we went around the main loop. Works great if main loop is faster than our response time.
- OTOH, our receiveByte() command was "blocking" – it just sat there and waited for data to come in.
- ```
  uint8_t receiveByte (void) {
  while (!(UCSR0A & _BV(RXC0)));/*Wait for data*/
  return UDR0; /* return register value */
  }
  ```
- What does the chip do if no serial data ever comes in?
- Wouldn't it be nice if we could do other stuff and then be interrupted when the receive-data register is full?
  ```

Event-Driven Programming

Brought to you by interrupts...

- Main loop: Does whatever it does, loops forever
- Interrupts handle the (infrequent) events, and then flow returns back to the main loop
- Further interrupts disabled while in ISR
- Interrupt requests are queued and there's a priority system, so if two other interrupts fire while you're in an ISR, it'll do the highest-priority one as soon as your ISR is finished.
- (You can enable other interrupts from within your ISR with `sei()` if you want them to be pre-emptable.)
- Eventually control returns to the main loop and continues
- How long does a cycle of the main loop take?

Event-Driven Programming

Two styles: lightweight or heavyweight ISRs

- Heavyweight version: When you press the button, the interrupt function writes out data to the serial line
- Lightweight version: Interrupt function sets a flag and returns. Main loop tests for this flag and handles the rest of the function if needed.
- Heavyweight is easier to code, immediate handling, but the ISRs block further interrupts – don't want to spend too much time here
- Lightweight spends less time in the interrupt routine, easier to handle prioritizing in mainloop, but the event doesn't run until you're back in the mainloop. May require global variables.

Outline

External Interrupts

Examples

- External interrupts are useful for syncronizing your chip's execution with the outside world
- For instance, I have an accelerometer that grounds a pin for 1ms every 10ms when it has a new data sample. Otherwise, it's connected to a pullup.
- One option is to poll that pin with the AVR, but it could miss it, or get out of sync, or....
- Better is have the hardware watch that pin and throw an interrupt when the pin is pulled low.
- In the next two examples, we'll use button presses, but keep in mind that the trigger could be any external action

External Interrupts

Code Example: Heavyweight

- Send light level over serial when press button
- Heavyweight: do ADC sampling and all within the ISR
- By default, interrupt is configured to trigger when the pin is low
- While holding low, ISR continues to run – stuck in ISR as long as hold down button
- Advantage: button-press is attended to exactly as it happens
- Disadvantage: spends a lot of time in the ISR – other interrupts will have to wait

External Interrupts

Code Example: Lightweight

- ISR just changes a variable to let the mainloop know
- Notice that variable is defined outside of the main loop so that the ISR (which is also outside the main loop) can access it
- Configure interrupt to fire only on falling-edge pin change
- If configured for low state as above, the ISR would continually re-write our variable's value, but only transmit data once you've let go of the button
- Advantage: easy to write debouncing, action handled in the normal mainloop, ISR doesn't block other interrupts
- Disadvantage: response is not instantaneous

External Interrupts

Summary

- Lightweight or heavyweight, here's what you need to do to use `INT0 and INT1`
- Enable the specific interrupt: `EIMSK |= _BV(INT0);`
- Make sure it's triggering on the right event: low, change, rising, falling
- Globally enable interrupts: `sei();`
- Write your ISRs (double-check the interrupt vector name!)

External Interrupts

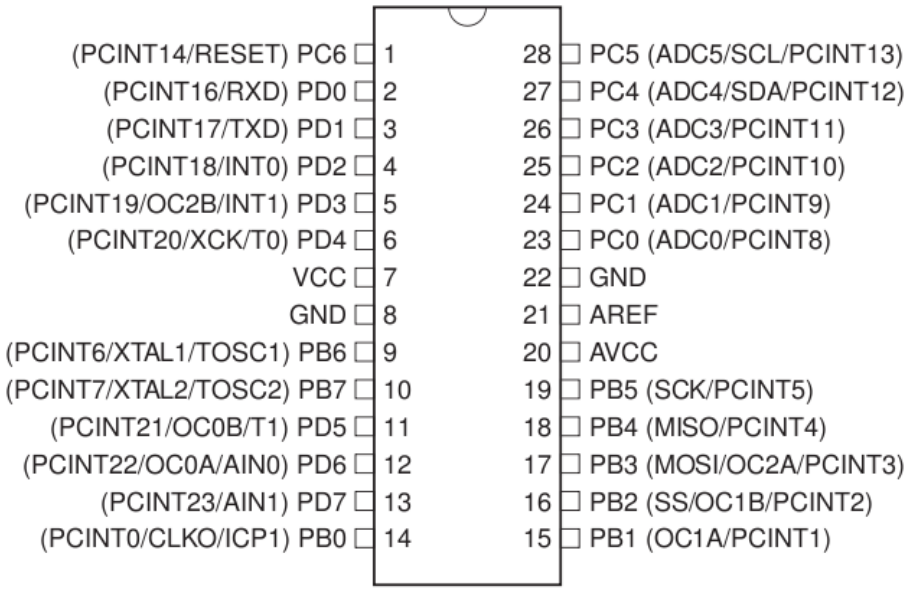
Misc Points

- You can use the light/heavy model to prioritize interrupts: heavies happen immediately and pre-empt, lights happen in the mainloop
- Can set `INT0, INT1` interrupts for rising, falling, toggle, or low state triggers
- The interrupt pins can still be used as output – can set an interrupt pin high in software to fire interrupts
- Debouncing becomes easier with timers, next class

Other Examples

- Many peripherals use external interrupts to signal when they've got data ready (GPS, accelerometer, SD cards, ...)
- IRQs on Compy386!

Pinouts



Pin-Change Interrupts

Interrupts on the Other Pins

- PD2 (our button pin) is special: INT0
- All the other pins have "pin-change interrupts" associated with them (p. 70)
- Pin change interrupts only tell you that a pin you're monitoring has changed state,
  you have to handle the rest of the logic in code
- Pin change interrupts are grouped in sets of eight:
  (PBn, PCn, PDn)
- You have to select which port you're interested in,
  then set a mask for which particular pins
- See datasheet, p. 71 and on

Pin-Change Interrupts

Configuration and Use

- Two-step config example: enable PC interrupt on PB6
- enable PORTB/PC0 interrupt: `PCICR |= _BV(PCIE0);`
- monitor PCINT6 via interrupt: `PCMSK0 |= _BV(PCINT6);`
- PC interrupts fire on *any change* in the state of *any* of the monitored pins
- In your `ISR(PCINT0_vect)`, you need to test which pin raised the interrupt using normal methods: `PINB & _BV(PB6)`
- Example in `pinChangeInterruptDemo.c`

Pin-Change Interrupts

Summary

- Unlike the special external interrupts, PC interrupts use aggregated interrupt vectors
- If any of the monitored pins in PB change state, the ISR is called
- You have to:
  enable the overall PC interrupt
  configure the pin mask to select which pins are monitored
  handle the remaining logic in your ISR
  globally enable interrupts: `sei();`
- Not bad for managing 23 potential interrupts!
- Troubleshooting: did you get the interrupt vector name right? Did you `sei()`?

Outline

Interrupt-driven USART

A Better Serial Reader

- Remember what we did before: blocking serial reads
- What happens if your robot loses serial connection?
- Better: have interrupt deal with incoming serial data
- Heavyweight: read & process the new serial data within the interrupt
- Lightweight: just store the data, set a flag indicating it needs processing
- Complications: the lightweight approach needs a buffer – a place to store the incoming serial data as it arrives

Interrupt-driven USART

Heavyweight Serial Example

- Mainloop transmits alphabet over serial
  (just to give it something to do)
- But it's interrupted every time we type something
- As before, define an ISR() and enable the interrupt call
- The Gotcha: Need to read in the UDR0 (serial data register) to clear the interrupt flag. Otherwise, it just keeps calling the interrupt routine. (p. 189)

Interrupt-driven USART

Lightweight Serial Example

- Mainloop transmits alphabet over serial, checks to see if it should advance LED
- Here the ISR just fills up a buffer with the incoming data
- Buffer lets you handle a bunch of data when you can
  (pretend that our mainloop took a while or something)
- Or you could handle it all at once
- For more interesting buffers, google "FIFO" "ring" or "circular buffer"

Interrupt-driven USART

Transmit Version?

- Sure!
- Imagine you were sending short bits of bursty data, too much for the serial baudrate
- Create an ISR that's triggered by the USART_TX interrupt, so that every time a transmission is complete, the next begins
- In ISR, check to see if your transmit buffer has entries. If it does, transmit them and advance the buffer.
- In your mainloop, you just need to put your data into the buffer – the ISR will take care of sending it along when it can

Interrupt-driven ADC

More of the same thing...

- Without belaboring the point, just like there are USART_RX and USART_TX vectors, there is an ADC interrupt vector
- If you want the maximum sample rate from the ADC, this is the way to get it
- Set up the ADC as before in free-running mode tied to an internal clock source
- `ISR(ADC){ ... }` and read in or process your data
- Bam!

Other Hardware Peripheral Interrupts

For later reference

- See p. 59 for list of all interrupt vectors
- See `iomx8.h` for their macro defines
- Whole bunch of timer-related interrupts – we'll get to these next class when we cover the hardware timers
- I2C and SPI data interrupts like USART_TX
- EEPROM write complete (it takes time)
- Even RESET works internally using its own interrupt (you can set this in software by writing low to PC6!)

Homework

Response-time Tester Game

- Start with the LEDs off
- Wait a while then flash the LEDs
- Using a for loop with a delay, start counting elapsed time
- When button pressed (use an interrupt) spit out the elapsed time over serial

Ghetto Oscope in "Ten" Lines of Code

- Every time you get a value on the ADC, send it out over serial
- This should be all configuration – your mainloop can be empty
- The rest is code on your computer.

Next Class

Timers!

- We've been doing a lot of `for` loops with delays in them
- Want a hardware-based way to count time or waste time
- Timers and counters!
- It's even better – the timers can fire off interrupts

The End

Outline