

“Weightless communications”

Duncan Louttit

MINOS 2009

Can you get useful real-time information from a mouse while it is moving at speed without adding ANY weight?

- Need data at 1ms/byte or better.
- No hardware changes to mouse (nearly).
- Mustn't take more than 50% of processor cycles.

It would be nice to get sensor readings from the mouse in real time. This should show how the maze walls affect the readings obtained. It should also show what the mouse sees as it comes to the end of a gap or past the end of a wall at speed.

It would be nice to get accelerometer readings during actual motion.

This idea started as using the infra-red comms. chips for TV remotes as cheap sensors for schools' micromouse.

They don't work as well as expected. There is a lot of processing inside the receiver and anything other than the specified data stream is filtered inside the chip.

Basically it only works with data packets about 40ms long and this is ridiculously slow.

But it made me think.

Thoughts:

Typical mice use LEDs and photodiodes or phototransistors to sense how much light is reflected from the maze walls. You can use these LEDs to send the data.

If you drive all the LEDs from the same processor pin, the data will be sent in lots of directions.

Some light comes out the sides of the LEDs vertically upwards so the receiver should have a good chance.

A frequency shift keyed system could be fast.

The frequencies can be generated using a timer in the mouse.

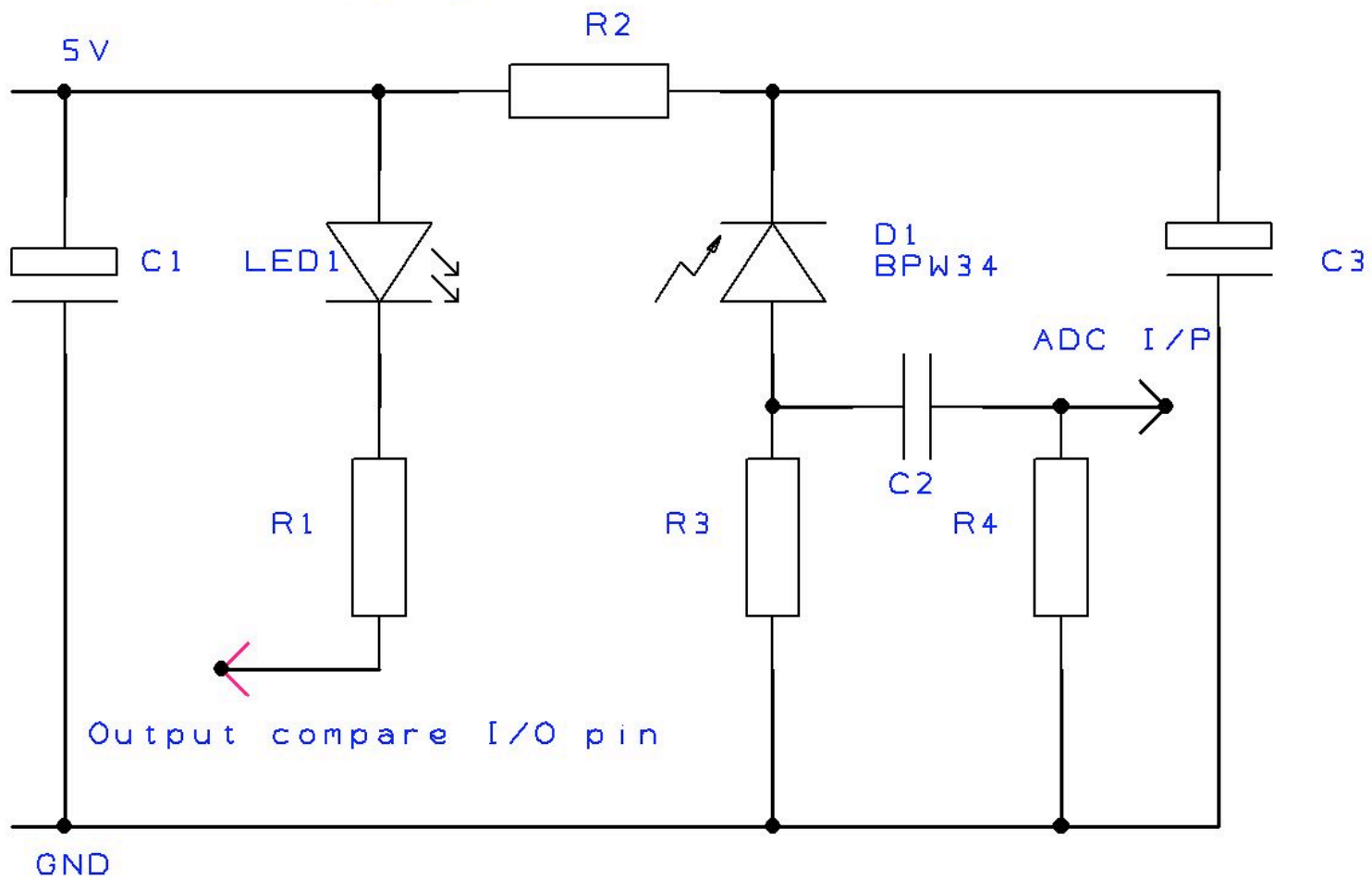
Drive the LEDs with the timer output and it will flicker suitably.

If the receiver is fast, you can send data in the gaps between sensor readings.

transmitter/sensor circuit

Infra-red communications TX version 1.0

Copyright 2009 D.R. Louttit



As a sensor it is pretty standard. It uses a photodiode for a fast linear response (less affected by ambient light).

This has a low output but the AVR I use has a X20 amplifier in the ADC system and a 1V reference so it only needs 50mV for full scale.

Unusually for me, the LED is an infra-red type. This gives maximum efficiency. It only needs 30mA of drive so can be driven directly from the pin. I would prefer visible light and I will try this at some future date.

For three sensors, use the LEDs in series if you have enough volts. Otherwise you will need to parallel them up, possibly with a buffer.

The scheme I use sends the data in UART format, LSB first at 9600 baud. All the timing and data formatting is done in software in the mouse.

Ones and zeroes are represented by frequencies near 160KHz.

One is 14 cycles of the lower frequency.

Zero is 17 cycles of the higher frequency.

Byte format is a start bit (0) followed by 8 data bits, followed by 2 stop bits (1). 11 bits altogether. At 9600 baud this takes 1.15milliseconds.

The transmit software is an interrupt driven “bit banging” UART that puts values into the timer compare register instead of driving an I/O port.

I use one timer. Interrupts are generated twice for each cycle of the output frequency. Every 14 or 17 cycles the interrupt has to decide if the frequency should be high or low.

The AVR is quite good at this sort of thing.

Transmitter interrupt listing.

```

TIMO-COMPA INTERRUPT OVERFLOW      \ 7 cycles max to here
IN  R15 , SREG                      \ 1 cycle

DEC  R9      \ Bit finished?      1 cycle
0< IF      \ 2 cycles if true
SEC        \ 1 cycle
ROR  R8      \ 1 cycle
CS IF     \ 2 cycles if true
MOV  R9 , R13 \ 1 cycle
OUT  OCR0A , R11 \ 1 cycle
ELSE     \ 2 cycles
MOV  R9 , R14 \ 1 cycle
OUT  OCR0A , R12 \ 1 cycle
THEN    \ 0 cycles !!
DEC  R10      \ 1 cycle
ELSE
OR   R10 , R10
0= IF      \ 2 cycles if true
OUT  TCCR0B , R10 \ Stop timer      1 cycle
OUT  TCCR0A , R10 \ Disable compare output to LED
OUT  TCNT0 , R10  \ Reset counter  1 cycle
THEN      \ 0 cycles!!
THEN      \ 0 cycles !!

OUT  SREG , R15 \ 1 cycle
RETI C;        \ 4 cycles to return

```

Slightly unusual listing; that's what FORTH assembler looks like!

R9 contains the number of interrupts before the bit finishes.

R8 contains the data to be sent

R10 contains the number of bits still to be sent.

R11 to R14 contain constants. R15 is used to hold the status register.

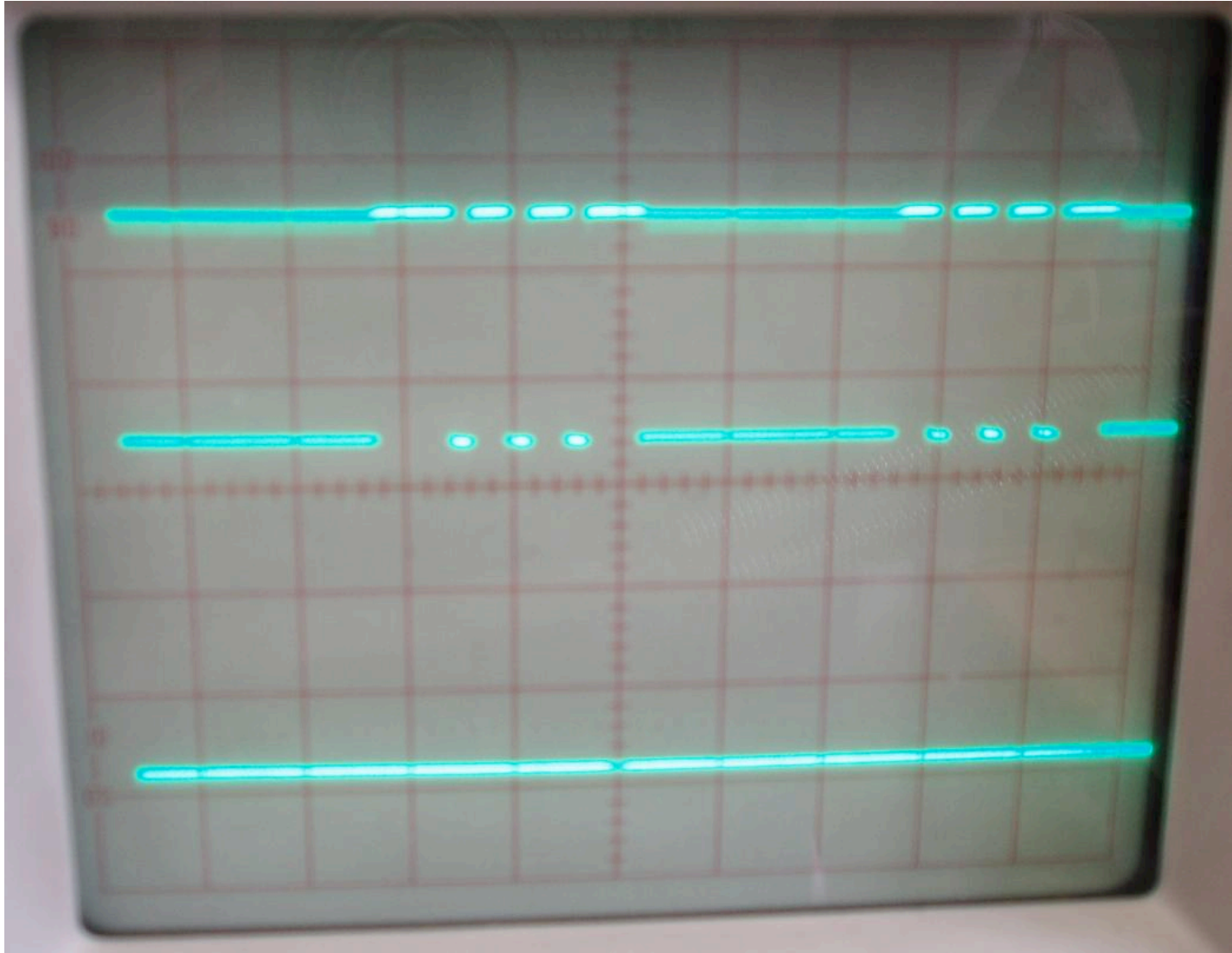
All the code is optimised for speed. With the internal 8Mhz oscillator there is not much spare time!

After a byte is sent, there needs to be a delay for the sensor on the mouse to recover. I use about 200 microseconds.

I pulse the LED 3 times, once for each sensor. I allow about 200 microseconds settling time between each pulse.

The complete sequence of sending a byte and reading 3 sensors takes about 2.5milliseconds. Fairly close to the target figure!

The AVR is running at 8MHz. It can run at 20MHz so there is room to halve the time needed.



Receiver

Needs to handle a wide dynamic range.

Needs to convert the signal into RS232 data for passing to a computer.

Simple is good!

Easy bits:

5V DC supply from a transformer/rectifier/regulator.

HIN232 TTL to RS232 level shifter.

In-system programming/debug facility; CONN1.

Harder bits:

Photodiode not phototransistor for speed.

2 tuned circuits for selecting the desired frequency. These are damped to prevent the data bits being “stretched” and ensure enough bandwidth for both frequencies.

Simple common-emitter gain stages.

“Clever” bits:

The signal goes into an analogue comparator in the AVR. This acts as a limiter. The information is held in the time between zero crossings. This only needs a few millivolts of signal.

The trimmer adds an adjustable D.C. offset to the signal. Signals less than the offset are not seen by the comparator. It is an analogue squelch.

Software

This uses the same timer as the transmitter. When the interrupt comes in, the software looks at the counter and then resets it ready for the next cycle. If the count is above a threshold the signal is a one, otherwise it is a zero.

Only change the output for two consecutive similar cycles to filter out noise.

The overflow interrupt on the same timer indicates that there is no signal and consequently the signal is a one.

Software

Output is setting/resetting an I/O port.

Because the timings are critical, the foreground task puts the processor to sleep between interrupts to ensure that the response times are consistent.

```

ANA-COMP INTERRUPT TRANSITION
    IN    R6 , SREG          \ 7 cycles to here
    PUSH R17                \ 1 cycle
                                \ 2 cycles

    IN    R17 , TCNT0       \ 1 cycle

    CLR   R5                 \ 1 cycle
    OUT   TCNT0 , R5        \ 1 cycle

    CPI   R17 , THRESHHOLD
    0> IF                    \ 2 if taken
        OR    R4 , R4       \ Last sample 1?
        0= NOT IF
            SBI   PORTB , 2 \ Output a 1. 2 cycles
            THEN
                INC  R5      \ Make a 1
                MOV  R4 , R5
            ELSE
                OR    R4 , R4 \ Last sample 0?
                0= IF
                    CBI   PORTB , 2 \ Output a 0. 2 cycles
                    THEN
                        MOV  R4 , R5
                    THEN

    POP   R17                \ 2 cycles
    OUT   SREG , R6         \ 1 cycle

    RETI  C;                \ 4 cycles to return

TIM0-OVF INTERRUPT RX_OVERFLOW \ If you get an overflow, it must be a 1
    IN    R5 , SREG

    SBI   PORTB , 2        \ Output a 1
    CLR   R4               \ Tell the filter the last bit was 1.
    INC   R4

    OUT   SREG , R5
    RETI  C;

CODE LOOPIT ( --- : Sleep between interrupts for fast response.)
    LDI   R16 , $20
    BEGIN
        OUT  MCUCR , R16
        SLEEP
    AGAIN

```

So, “weightless” communication is possible.

“Waitless” communication is more of a problem.

Things I learnt:

Putting the photodiode/phototransistor between load and supply is better than putting it between load and ground.

HIN232s generate a load of rubbish at about 125KHz.

Sharp infra-red detectors for TV remotes are pretty useless for micromouse.

Hyperterminal doesn't work well on my dual core PC. I ended up downloading Realterm instead.

There's quite a lot of jitter on the sensor readings. This seems to be due to ambient light interference.