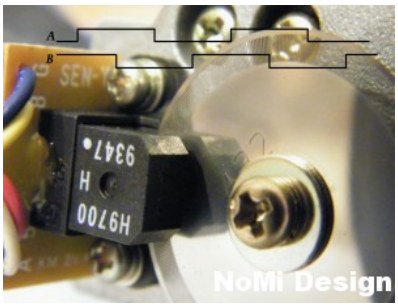
NoMi DESIGN
Making good things happen...

Inline Assembly: A Fast Quadrature Decoder…

Posted on December 21, 2011 by krazatchu

Inline assembly is a great tool to have in your programming arsenal. It gets things done as fast as a *kitten chasing a hummingbird*, yet retains the usability and portability of C code.



This bit will decode a quadrature input from a rotary incremental or linear position sensor, it polls two inputs (PIND4 & PIND5) to convert the incoming grey code into respective direction and count, tallying the position in a signed 16bit variable. It can be run on pinchange interrupt, timer interrupt, or called as a function. It also contains a provision for missed transitions.

Globally we have three declared variables, the current position and a couple of previous variables. As they will be modified inside the inline statement, we have declared them as volatile:

```
volatile int CurrentPosition;
volatile unsigned char PreviousEncoder;
volatile unsigned char PreviousDirection;
```

We also have a locally declared 16 bit variable, used for temporary storage during calculations. In assembly this would normally be handled by a pair of the x, y, z doubles:

```
unsigned int Zbuffer = 0; // setup a 16bit temporary local buffer
```

In standard inline form, the inputs and outputs are listed as the last bit, separated by colons. As zbuffer is a temporary variable it's only declared in the output, but must be constrained to upper registers (r16 to r31) as it will be used with immediates. The input uses the integer constraint while everything else is relegated to any available register. This allows for the compiler to have the most flexibility in assigning resources:

```
50  // Output
51  :  "=r" (CurrentPosition),
52     "=d" (Zbuffer),
53     "=r" (PreviousEncoder),
54     "=r" (PreviousDirection)
55
56  // Input
57  :  "I" (_SFR_IO_ADDR(PIND)),
58     "0" (CurrentPosition),
59     "2" (PreviousEncoder),
60     "3" (PreviousDirection)
```

The original assembly was written by Chan of ElmChan, it was converted to inline and republished with permission under GPL. This code has been tested on a DC Servo sporting a Mega8 @ 16Mhz, running inside a timer interrupt at 100 kHz. No registers were harmed in the making of this code. Finally without further NOPs, here is the complete code:

```
1  unsigned int Zbuffer = 0; // setup a 16 bit temporary local buffer
2
3  asm volatile(
4  // incoming stack maintance is handled by compiler
5
6  "mov    %A1, %2     \n\t"    // Store previous encoder in low buffer
7  "in     %2,  %4     \n\t"    // Input encoder state PORTD 0b00xx0000
8  "swap   %2          \n\t"    // Swap nibbles 0b76xx3210 -> 0b321076xx
9
```

```
10  "ldi    %B1, 1      \n\t"    // Load 1 into high buffer
11  "sbrc   %2,  1      \n\t"    // Skip next instruction if bit 1 is clear
12  "eor    %2,  %B1    \n\t"    // Exclusive OR, ENC = 1x, if 10 -> 11, if 11 -> 10
13
14  "sub    %A1, %2     \n\t"    // Subract previous from 11 or 10, store in low
15  "andi   %A1, 3      \n\t"    // AND with 11 - check if bits have changed
16  "breq   exitpoint   \n\t"    // If equal, no change, jump to exit
17
18  "cpi    %A1, 3      \n\t"    // Compare low buffer to 0b00000011
19  "breq   decrement   \n\t"    // If equal jump to decrement
20
21  "cpi    %A1, 1      \n\t"    // Compare low buffer to 0b00000001
22  "breq   increment   \n\t"    // If equal jump to increment
23
24  "mov    %A1, %3     \n\t"    // No branch = lost in transistion
25  "mov    %B1, %3     \n\t"    // Use previous direction to make up lost bit
26  "lsl    %A1         \n\t"    // Logical left shift = x2 - double up for lost bits
27  "asr    %B1         \n\t"    // Clear and save signed flag
28  "rjmp   summation   \n\t"    // Jump to total
29
30  "decrement:         \n\t"
31  "ldi     %A1, -1    \n\t"    // Load low buffer with -1
32  "ser     %B1        \n\t"    // Set high buffer for negative number = 0xFF
33  "rjmp storeprevious \n\t"    // Jump over/skip increment
34
35  "increment:         \n\t"
36  "ldi     %A1, 1     \n\t"    // Load low buffer with 1
37  "clr     %B1        \n\t"    // Clear high buffer
38
39  "storeprevious:     \n\t"
40  "mov     %3,  %A1   \n\t"    // Store the previous direction in low buffer
41
42  "summation:         \n\t"
43  "add     %A0, %A1   \n\t"    // Add low byte and carry to high
44  "adc     %B0, %B1   \n\t"    // This becomes the 16bit variable CurrentPosition
45
46  "exitpoint:         \n\t"    // final destination
47
48  // outoging stack maintance is handled by compiler....
49
50  // Output
51  :  "=r" (CurrentPosition),
52     "=d" (Zbuffer),
53     "=r" (PreviousEncoder),
54     "=r" (PreviousDirection)
55
56  // Input
57  :  "I" (_SFR_IO_ADDR(PIND)),
58     "0" (CurrentPosition),
59     "2" (PreviousEncoder),
60     "3" (PreviousDirection)
61
62  // Clobber
63  :  // nothing to clobber
64
65  );
```

An example application and the current test platform is the DC Servo kit...

This entry was posted in ASM, AVR, Embedded, Encoder, Motion Control, Sensors. Bookmark the permalink.
