

Introducing MC9S12DT256 port of StickOS BASIC (beta, long)...

Hi all,

Recently, I had a few embedded system projects I wanted to do, including controlling my toaster oven to do SMT reflow soldering... In addition to that, one of my dreams for a long time has been to make these kinds of projects more accessible to high-school and college age hobbyists, like I believe they used to be in the past.

Anyway, one thing led to another and we ended up writing an entire MCU-resident “StickOS BASIC” interactive embedded system programming environment *that runs entirely within the MC9S12DT256 MCU*, including an easy-to-use editor, transparent line-by-line compiler, interactive debugger, and flash filesystem, where external MCU pins can be mapped to special “pin variables” for manipulation or examination, and internal MCU peripherals are managed by BASIC control statements and BASIC interrupt handlers.

In literally minutes, folks can be interactively using most of the components and capabilities of the MC9S12DT256 MCU, including:

- digital I/O pins, via the *Port Integration and Multiplexed External Bus Interface Modules*
- analog input pins, via the *Analog-to-Digital Converter Module*
- analog output (PWM) pins, via the *Pulse-Width Modulator Module*
- frequency output pins, via the *Enhanced Capture Timer Module*
- UART I/O pins, via the *Serial Communications Interface Modules*
- interval timers, via the *Clock and Reset Generator Module*
- non-volatile storage, via the *Flash Memory Module*

If anyone wants to try it out, StickOS BASIC runs as-is on a APS12DT256SLK Board (or basically any board using a MC9S12DT256 MCU). Download information, including the .elf HCS12 binary image and full documentation, is on the web at: <http://www.cpushick.com/downloads.htm>

Once StickOS has been flashed to the MCU, you can log into the MCU’s TXD0/RXD0 (PS1/PS0) pins from any host computer with a serial port set to 9600 baud, 8 data bits, no parity, and xon/xoff flow control and then control it by any terminal emulator program, *with no additional software or hardware required on the host computer.*

Note that if you are talking to a bare MCU, you can use a RS232 Shifter Board Kit (such as http://www.sparkfun.com/commerce/product_info.php?products_id=133) to convert the MCU pin levels to RS232 used on your host computer.

By its very nature, StickOS supports in-circuit emulation when it is running in the MC9S12DT256 -- all you need is three serial port wires connecting the MCU to a host computer, and you have full control over the target embedded system! The host

computer may then be disconnected for standalone operation. (Or it may be left attached if you just want to use the MC9S12DT256 as a slave data acquisition device!)

When you initially log in you'll see a command prompt in the terminal emulator window like:

```
Welcome to StickOS for Freescale MC9S12DT256 v1.50i!  
Copyright (c) 2008; all rights reserved.  
info: http://www.cpustick.com  
bugs: support@cpustick.com  
(checksum 0x0)  
> _
```

If anyone remembers the old HP-2000 TimeShare BASIC systems from the 1970's, that's what inspired me to do this... In StickOS BASIC, however, the CPU and filesystem, not to mention the I/O system, are all on a single chip! You talk directly from the terminal emulator to the MC9S12DT256!

Hello World!

So what does the "Hello world!" program look like in StickOS BASIC?

Well, if the baseline goal for an embedded system is to configure an I/O pin and get an LED to blink, such as the LED3 on pin pb6 of the APS12DT256SLK Board, then the "Hello world!" program looks like this (entered text is in **bold**):

```
> 10 dim led3 as pin pb6 for digital output  
> 20 while 1 do  
> 30   let led3 = !led3  
> 40   sleep 500 ms  
> 50 endwhile  
> run  
<Ctrl-C>  
STOP at line 40!  
> _
```

Line 10 declares a "pin variable" named "led3", then configures the general purpose I/O pin "pb6" for digital output, and finally binds the pin variable to the corresponding pin (in traditional BASIC, the "dim" statement is used to "dimension" the shape of a variable prior to use). From then on, any modification of the pin variable is immediately reflected at the I/O pin. Line 20 starts an infinite loop. Line 30 inverts the state of the pb6 digital output pin. Line 40 delays the program for 500 ms. And finally line 50 ends the infinite loop. Press <Ctrl-C> to stop the program.

Of course, if you really *just* wanted to print "Hello world!" to the terminal (assuming it was still connected :-), you could just do:

```
> 10 print "Hello world!"
> run
Hello world!
> _
```

Hello Potentiometer!!

If you want to read analog input pins, or set analog output pins (PWM actually), in StickOS BASIC it's just as easy...

If you have a potentiometer connected to pin pad00 on the APS12DT256SLK Board, you can read it with:

```
> 10 dim pot as pin pad00 for analog input
> 20 while 1 do
> 30   print "pot =", pot
> 40   sleep 1 s
> 50 endwhile
> run
pot = 2117
pot = 2117
pot = 2117
pot = 2117
<Ctrl-C>
STOP at line 120!
> _
```

Line 10 declares a “pin variable” named “pot”, then configures I/O pin “pad00” for analog input, and finally bind the pin variable to the corresponding pin. From then on, examination of the pin variables results in the current ADC values being read, in millivolts (mV). Lines 20-50 poll the potentiometer every second and print the results to the terminal, in millivolts (mV). Press <Ctrl-C> to stop the program.

Hello Toaster Oven!!!

What if you want to use interrupts? StickOS BASIC supports them also! You can configure timers to interrupt periodically and UARTs to interrupt on character transmit or receive. You can also configure any input pins to interrupt thru their use in a watchpoint expression.

The following program illustrates the use of a timer interrupt (at lines 50-60 and 130-190), and controls my toaster oven with a convenient SMT reflow temperature profile, when hooked to a K-type thermocouple thru a 100:1 op-amp!

```
> 10 dim target, secs
> 20 dim thermocouple as pin pad00 for analog input
```

```

> 30 dim relay as pin pa0 for digital output
> 40 data 512, 90, 746, 105, 894, 20, -1, -1
> 50 configure timer 0 for 1 s
> 60 on timer 0 do gosub adjust
> 70 while target!=-1 do
> 80   sleep secs s
> 90   read target, secs
> 100 endwhile
> 110 let relay = 0
> 120 end
> 130 sub adjust
> 140   if thermocouple>=target then
> 150     let relay = 0
> 160   else
> 170     let relay = 1
> 180   endif
> 190 endsub
> save
> autorun on
> -

```

Note that if terse code were our goal, lines 60 and 130-190 could have all been replaced with the single statement:

```

> 60 on timer 0 do let relay = thermocouple<target

```

“Save” saves the program to the non-volatile flash filesystem; “autorun on” sets the program to run automatically when the MCU is powered up.

Hello Dolly???

Using frequency output pins bound to the output compare timers whose value can be set in hertz (Hz), you can play scales or music or whatever else you might dream up, such as:

```

> 10 dim oct
> 20 dim freq as pin pt0 for frequency output
> 30 let freq = 440
> 40 for oct = 1 to 8
> 50   sleep 500 ms
> 60   let freq = freq*105946/100000
> 70   if oct!=3&&oct!=7 then
> 80     let freq = freq*105946/100000
> 90   endif
> 100 next
> 110 let freq = 0

```

Which plays a single octave of the musical scale starting at A-440 (Hz) on the pt0 (IOC0) output pin.

Zigbee Support!

When coupled with an optional Freescale MC13201 Zigbee Wireless Transceiver (such as in the 1320xRFC daughter card), the MCU may be remotely controlled by another MCU, via a telnet/rlogin-like interface, eliminating the need for a direct connection to the host computer altogether.

Additionally, BASIC programs may trivially remotely access variables on other MCUs, enabling the use of “remote pin variables” or other forms of inter-MCU communication.

See <http://www.cpushick.com/wireless.htm> for details, and an example of how trivially easy it is to build a wireless embedded system, like a remote LED dimmer.

Other StickOS features...

- * on-line help
- * ansi terminal line editing
- * transparent line-by-line compilation and de-compilation and bytecode execution
- * full-featured debugger with breakpoints, watchpoints, assertions, and even edit-and-continue
- * load and save up to three BASIC programs in flash
- * prolong flash lifetime by storing incremental updates in RAM

Who is StickOS for?

Our main target audience for StickOS is folks doing low-volume hobbyist and educational work. Or more specifically, our main target audience is folks for whom “ease of development” (and even ease of re-development!) is significantly more important than “production cost”. Obviously, for hobbyist and educational use, there is no production!

Who is StickOS **not for?**

StickOS simply doesn't make sense for high-complexity, high-performance, high-volume, or cost-sensitive applications. By pushing the development environment to the MCU, you end up using a \$3 MCU where you could have gotten by with a \$1 MCU, from a performance and resource usage perspective. Likewise for high-volume applications, development costs (and “ease of development”) are nearly moot, since they are amortized over large number of production units.

Where is more information on StickOS?

See <http://www.cpushick.com> . You can download the .elf HCS08 binary image to flash to your APS12DT256SLK Board, as well as a full set of documentation. There are also C source code examples for the “general purpose” parts of StickOS I thought folks would find useful, including the serial transport device driver, I/O pin manipulation, flash programming, etc. These source files are available with no restrictions; however, we ask you don't republish them so we can fix bugs in the originals.

-- Rich