BLE Software Spec

Version 2.0
Pat Walp, KSI -- 2023 August 8

Introduction:

This document is my concept of software that will allow us to use the CYBLE in our system. Ideally Arrow/Infineon would develop all of this, leaving very minor tweaks for us to do. In the event that this is too much, the more important part is the software running on CYBLE. Please have a look at this, and then let's talk about it.

The level of detail here is intended to be a balance of full "specification" and reasonable brevity. I can think of a lot of things that should be added to specify things completely, but for the most part these are things I don't really care about. Questions and discussion are welcome.

The PC Operating System is assumed to be Windows, and the PC source is assumed to be Python. Also assuming that the Python is compiled to a Windows-native .exe file. (Background: We may have an interest in a Linux executable at some point, but this only something to be aware of, not necessarily to put any effort into.) Let us know if you would prefer otherwise.

Two Components:

1. Running on CYBLE
    a. Carry out commands from host
    b. Collect data from 18 bits of digital inputs (GPIO)
    c. Talks to 8 i2c slaves within target
    d. Talks to FPGA UART within target
2. Running on PC ("host")
    a. Accepts commands from command line
        i. Returns results of those commands in the same terminal. All output goes to stdout, with an option to be copied to a text file.
    b. Provides option to accept input from
        i. standard input
        ii. a text file
    c. Command-line editing: The usual
        i. up/down arrows for command history
        ii. left/right arrows to move cursor within a command
        iii. type to insert, backspace to delete

Host (PC) User Interface

This is a command line application, compiled to an executable "dble.exe". It is executed within a DOS window by typing "dble". Within the application each command is entered on a command line. Output is displayed immediately after. Output from commands that interact with the target (other than uart) start with a time stamp. If invoked with a single parameter, take that parameter to be a script file. (See the **`infile`** command below.)

Commands:

rb GPIO_SPEC -- read bits and display

GPIO_SPEC is a list of ports, e.g., "P5.0,P0.4,P6.7". Separator is either comma or space.

Output is a single line of text, starting with a time stamp in milliseconds, followed by a bit vector with the states of the bits in GPIO_SPEC, terminated with a newline character. For example: "1,234.560ms 10_1110\n". Long bit vectors have "_" inserted every four characters for readability. In this example, six bits were read.

rbl -- repeat the last **rb** command in a loop. Ctrl-C stops.

lp PERIOD -- set loop period in milliseconds. If an iteration takes more than **lp** milliseconds, execute the next iteration immediately, prepending its output with "*". Applies to all loop commands (rbl, ril, wil). Before the first **lp**, PERIOD is 500.

bin -- set the radix to binary (for **rb** command). This is the default.

hex -- set the radix to hexadecimal (for **rb** command)

toff/**ton** -- turn time stamp display off/on for future commands. Initialize to on.

ri BUS ADDR -- read from an i2c register. BUS is a number from 0 to 7 specifying the i2c switch port. ADDR is the 7-bit i2c address (hexadecimal).

wi BUS ADDR DATA -- write to an i2c register. BUS is a number from 0 to 7 specifying the i2c switch port. ADDR is the 7-bit i2c address (hexadecimal). DATA is the data to be written (hexadecimal). (Optional: If it's easy to do, accept multiple DATA, incrementing the ADDR for each write. This is common i2c practice.)

ril -- repeat the last **ri** command in a loop. Ctrl-C stops.

wil -- repeat the last **wi** command in a loop. Ctrl-C stops.

rt -- reset time to 0, and don't restart it until the next loop command (**rbl**, **ril**, **wil**). (Time is also reset to 0 when dble starts.)

reset_fpga -- drive P0.4 low for about 10 ms, then undrive it (leave it high-Z)

reset_switch -- drive P9.2 low for about 10 ms, then drive it high (leave it high)

uart -- change to UART mode, during which keyboard strokes are echoed to the FPGA UART, and characters received from that UART are echoed to the output. Ctrl-alt-C exits this mode. Each character is sent (to the FPGA) as soon as the key is pressed, and each character received (from the FPGA) is immediately displayed. Format is 115200 8N1.

logfile FILENAME -- start logging output to text file. If FILENAME is omitted, open a dialog box to choose location and file name. If FILENAME is relative, use the same folder as last time, defaulting to user's home.

logstop -- stop logging output to text file and close the file.

infile FILENAME -- execute all the commands in text file script. If FILENAME is relative, use the same folder as last time, defaulting to user's home.

exit -- exit dble. (Note that when executing **infile**, at the end of the file revert to interactive mode, unless there is an **exit** within the script.)

help -- print a summary of commands (maybe just condensed version of this "spec")

Initialization:

Initialize all GPIOs not labelled in the schematic for special purposes to be inputs.

Initialize P0.4 (PS_POR_B) to be hi-Z.

Initialize P9.2 (SWITCH_RESET_B) to 1.

PERIOD = 500 (milliseconds)

radix = bin

timestamp = 0 (and does not start incrementing until first loop command)

timestamp display = on

output logging = off

logfile and infile folder = user's home

Notes:

Timestamp resolution: Assumed to be a small number of microseconds. This is not a requirement. If 10 milliseconds is the best we can do, that will be adequate. We would like the highest resolution that is reasonably easy to achieve.

i2c Addresses are in 7-bit format.

Error handling:

Invalid user input generates a useful diagnostic.

- BUS is a single digit 0..7
- ADDR is a 1- or 2- character hexadecimal number 0-7e (i2c address in 7-bit format)
- DATA is a 1- or 2-character hexadecimal number 0-ff
- In the **rb** command, only the ports with nets labelled generically in the schematic (e.g., "P10.3") are accepted. An exception: P0.4 (net name PS_POR_B) can be read.
- All user input is case insensitive

i2c NACK errors print one of the following:

- i2c read NACK bus=n addr=0xdd
- i2c write NACK bus=n addr=0xdd data=0xdd

Appendix: Suggested implementation of loop timing

The timestamp output at the beginning of each iteration indicates when the iteration starts. If LoopPeriod (as set by the "lp" command) is larger than the execution time of one iteration, then the resulting series of timestamps should be spaced by that period (or as close to this as possible). There is more than one way of implementing this, but here is one possibility:

In the following, "current time" is assumed to be retrieved from the PC OS as an integer number of microseconds (maybe with Python "microsecondsSinceEpoch", or something like that). Time stamps are calculated from the difference between the current time and the time of the last "rt" command.

When one of the loop commands is invoked ("rbl", "ril", "wil"), set CurrentTime to the current time and NextTime to CurrentTime + LoopPeriod. Output CurrentTime (in the format shown above). Execute the command to be iterated ("rb", "ri", or "wi") once. Then iterate as follows:

1. Begin Iteration: Set CurrentTime to the current time. Depending on whether NextTime has passed, do one of the following:

2. If CurrentTime > NextTime:
    a. Set NextTime to CurrentTime + LoopPeriod.
    b. Output "*", CurrentTime, and then execute the command to be iterated.

3. If CurrentTime <= NextTime:
    a. Wait until CurrentTime => NextTime.
    (Maybe by setting a timer to continue when the current time reaches NextTime, or something like Python "time.sleep((NextTime-CurrentTime)/10^6)")
    b. Set NextTime to NextTime + LoopPeriod
    c. Output CurrentTime, then execute the command to be iterated.

Note that during the looping, Ctrl-C exits, and spacebar pauses/unpauses. Checking for these two inputs should happen just once per loop, at the top of each iteration (labelled "Begin Iteration" in the above).