Embedded systems – Exercise #1

1 Exercise purpose

In this exercise you will build a device with basic Input / Output system. You will learn to write drivers for hardware devices.

You will need to write the code that operates this device, including:

- Drivers for the hardware devices (clock, display, and serial port)
- Code for the input/output protocol.

You are required to follow the submission guidelines from the course's web site.

2 Device description

The embedded device you will build has a display that shows the value of a counter C. C's initial value is 0 and its value is increased by 1 every T milliseconds (on your simulated environment it will actually be ~10 times slower). The default value of T is 100. C is 8 bits wide and when it passes 255 it wraps around (i.e. the value after 255 is 0). T is also 8 bits wide ($0 < T <= 255$)

The device also has a serial input/output port. The port allows reading and writing the values of the counter and T, and resetting the system. This is done using a simple protocol.

3 Device specification

The device has 8KB of ROM starting at address 0x0, and 48KB of RAM starting at address 0x4000. The device contains three hardware devices: clock, 7-segments display, and serial port. Memory mapping and the specification of the device registers can be found in the Device Specification Guide.

In order for the simulator to "have" these Hardware devices and memory layout, you need to download the ".so" files from the website and make sure the simulator "connects" to them. It also has to be configured to "have" this memory layout. This can be achieved either by manually configuring the GUI or using an argument file when running the simulator (the latter is more recommended).

You will get a memory configuration file (linker script). The file will be in SVR3 Command-File Format. You can find a reference of the SVR3 format in the "ARC ELF Linker and Utilities" guide, chapter 5.

4 Device Drivers

Your main program must not access the hardware peripherals directly, so you are required to write and use device drivers. You will need to provide drivers for all three hardware devices. Design a good API for each driver and then use this API in the "main" program. All the drivers you will write in the exercise will be polling-based and will not use interrupts.

5 Protocol specification

What is a protocol? It is an agreed communication method which defines how and when to send each component, what does it contain and what is its structure, how to interpret it, etc.

Each command in our protocol is comprised of a request (sent to the device) and a response (sent from the device). Commands are being sent to our embedded system via the UART. The system parses the command and acts accordingly.

Each request or response is 4 bytes long, and is comprised of ASCII characters.

Request structure:

Byte 1: command byte. Either ‘r’ for “read” or ‘w’ for “write”. Have to be lowercase.

Byte 2: variable byte. Either ‘c’ for the counter C or ‘t’ for the interval T. Have to be lowercase.

Bytes 3-4: value bytes. 2 hexadecimal digits range from 00 to FF. On “Read” requests those bytes have to be “00”. On “Write” requests those bytes designate the current value of the variable. This value can be either lower or upper case.

Response structure:

Byte 1: command byte. Either ‘R’ for “read” or ‘W’ for “write”. Have to be uppercase.

Byte 2: variable byte. Either ‘C’ for the counter C or ‘T’ for the interval T. Have to be uppercase.

Bytes 3-4: value bytes. 2 hexadecimal digits range from 00 to FF. On “Write” responses those bytes have to be “00”. On “Read” responses those bytes designate the current value of the variable. This value must be in upper case.

The 4 allowed commands are (VV stands for a value, designated by 2 hexadecimal digits)

1. Read Interval:
    a. Request: rt00
    b. Response: RTVV
2. Write interval:

a. Request: wtVV (Note: VV cannot be 00)

b. Response: WT00

3. Read Counter

    a. Request: rc00

    b. Response: RCVV

4. Write Counter:

    a. Request: wcVV

    b. Response: WC00

Errors and synchronization:

1. Error:

    If the device could not parse a request, or if the values are incorrect, the device returns as a response the string "Z".

    The device shall send the "Z" string only after receiving the full four characters of the command, even if the error can be recognized before that.

2. Reset:

    If the device gets at any point in time the string "X" it should flush any command that it is parsing and be ready to receive a new command. In any case it should not send any new response.

The protocol is synchronous. This means that the sender will wait for a response before sending a new command (except for reset which does not send a response).

Bonus (Max 10 points): Comply with an asynchronous protocol. The client is allowed to send multiple requests without waiting for a response between the requests. The requests must be served in order, and the responses must be transmitted in the same order of the requests.