Embedded systems – Exercise #2

1 Exercise purpose

In this exercise you will implement several device drivers for a complex device: 7-segments, serial, timer, LCD display, input panel, and flash.

You will implement a debug protocol through the device's serial port to control the devices and retrieve status.

You will get familiar with interrupts.

2 Device description

The device has 16KB of ROM starting at address 0x0 and 48KB of RAM starting at address 0x6000. The device contains six hardware devices: timer, 7-segments display, LCD display, input panel, flash, and serial port. Use ex2.args as the simulator arguments file. Memory mapping and the specification of the device registers can be found in the Device Specification Guide

Note: the extension file embsys_pager.so implements both the input panel and the LCD screen in one cellular-phone-like device.

You will need to write and supply the linker with a memory configuration file. 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.

3 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 six hardware devices. You will need to implement interrupt handlers, when applicable.

4 Protocol Specification

The protocol is a debug protocol that allows a debugger to control the peripherals and receive status through the device's serial port. Note that the "debugger" is not the one you develop with – MDB. The meaning is that a conceptual device, a debugger in this example, can connect to your embedded system and work with it. A program such as embsys_term can assume the role of the debugger, sending and receiving command through serial.

The protocol is message based. There are messages sent from the debugger to the device, and messages sent from the device to the debugger. In each direction, only one message is sent at a

time. However, messages from the device to the debugger are asynchronous and should not interfere the messages in the other direction, or be interfered by them.

Inbound requests must be executed in the order they arrive. However, if the device cannot execute a request because some resource is busy, it needs to queue the request, continue to serve other requests, and as soon as that resource becomes available execute the original request. You only need to queue up to 5 requests, and response with an error message to excess requests that need to be stalled. (Consider the case where 2 requests for a resource A immediately followed by 2 requests for a resource B, arrive at the device. The device can execute the 1st request, but needs to queue the 2nd. Then it can execute the 3rd, and needs to queue the 4th.)

Each message starts with two English uppercase characters specifying the message type. They are called the "message code".

Next are the message fields. The fields are sent in ASCII hexadecimal representation, so there are two characters sent for every byte in a field.

All messages ends with a period character ('.', ASCII 0x2E). This allows the size of the last field in each message to be of undefined length. The device will process the message only when the period character is received.

When the device receives a malformed message, it responds with an error message ("ZZ."). Errors can arise from various parsing errors, such as unidentified command, incorrect character, unexpected length, or errors in the semantic level, such as attempting to read/write from an invalid flash address.

4.1 Messages from the debugger to the device

Read Flash (Request)

Field	Length	Notes
Message Code		"RF"
Start Address	2 bytes	
Size	2 bytes	0-512 bytes
End Marker	1 byte	0x2E

Write Flash

Field	Length	Notes
Message Code		"WF"
Start Address	2 bytes	
Buffer	<undefined>	0-512 bytes
End Marker	1 byte	0x2E

Flash Bulk Erase

Field	Length	Notes
Message Code		"EF"
End Marker	1 byte	0x2E

Flash Block Erase

Field	Length	Notes
Message Code		"EB"
Start Address	2 bytes	
End Marker	1 byte	0x2E

Seven Segments

Field	Length	Notes
Message Code		"SS"
Most Significant Byte	1 byte	Bitmask of which LEDs to set on the most significant byte, see device specification for details
Least Significant Byte	1 byte	Bitmask of which LEDs to set on the least significant byte, see device specification for details
End Marker	1 byte	0x2E

Button Interrupts control

Field	Length	Notes
Message Code		"BI"
Enable/Disable	1 byte	0 – disable interrupts (default) 1 – enable interrupts
End Marker	1 byte	0x2E

Set Timer

Field	Length	Notes
Message Code		"ST"
Timer value	2 bytes	Requested expiration interval from now, in milliseconds. Message overrides any other timer message previously set.

Periodic	1 byte	0 – non-periodic timer 1 – periodic timer
End Marker	1 byte	0x2E

Display Message

Field	Length	Notes
Message Code		"DM"
Row Number	1 byte	0-16
Selected	1 byte	0 – display text as not selected 1 – display text as selected
Buffer	<undefined>	Up to 11 characters
End Marker	1 byte	0x2E

4.2 Messages from the device to the debugger

Error

Field	Length	Notes
Message Code		"ZZ"
End Marker	1 byte	0x2E

Read Flash (Response)

Field	Length	Notes
Message Code		"RF"
Buffer	<undefined>	
End Marker	1 byte	0x2E

Button Pressed

Field	Length	Notes
Message Code		"BP"
Button Pressed	1 byte	Value 0-C, per device specification
End Marker	1 byte	0x2E

Timer Expired

Field	Length	Notes
Message Code		“TE”
End Marker	1 byte	0x2E