Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

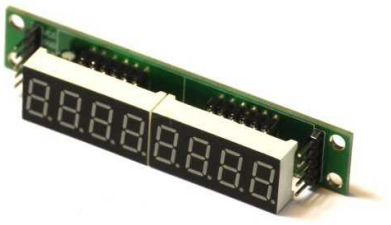
System Construction Course 2018,

Assignment 5

Felix Friedrich, ETH Zürich

Introduction

In this exercise you will implement an SPI device driver and the command interface in order to interface with an 8-digit 7-segment display that is driven by a MAX7219 display chip with SPI interface.



Lessons to Learn

- How to implement an SPI device driver.
- How to communicate with a particular SPI device.

Preparation

1. Open a console in directory assignments/assignment5
2. Make sure you have a proper kernel running on your RPI. If you need to recompile the kernel, you can do so by calling `oberon execute MakeMinos.txt`. You can then use module loading and are not required to link the kernel again for the rest of this exercise.

1 SPI

1. Consult the BCM2835 manual (page 102) together with the RPI pinout in order to find out how to connect the 8-digits 7-segment display to the RPI. We will use the connectors associated with device SPI0. The corresponding connectors on the MAX7219-based display are labeled as VCC (Voltage 3.3 V), GND (Ground), DIN (Data In), CS (Select) and CLK (Clock).
2. Browse Chapter 10 (SPI, p. 148) of the BCM2835 manual: Read Sections 10.1, 10.2 (omitting 10.2.2) and browse through 10.4, 10.5 and 10.6 superficially.
3. Write the driver module `SPI.Mos`
   (a) `SPI.SetGPIOs`: Setup the GPIO pins correctly. Code patterns to access bits are described below on Page 3. Make use of function `Platform.ClearAndSetBits` in order to set alternate function 0 for pins 7 – 11.
   (b) `SPI.Init`: Setup a reasonable SPI clock frequency (not more than 10 MHz) by writing to the `SPI_CLK` register. The APB clock runs at 250 MHz. Setup phase and polarity correctly according to the pin description table on page five of the manual of the MAX7219 display chip. Reset the buffers by setting the `CLEAR_RX` and `CLEAR_TX` bits in the `SPI_CS` register.
   (c) `SPI.Write` Implement this procedure to send an array of 8-bit characters via SPI. The steps to drive the SPI interface in a polled way are described in 10.6.1 of the BCM 2835 manual as follows:
      i. Set `TA` bit of `SPI_CS` register. This will pull chip select low.
      ii. For each character do:
      Poll `TXD` or `SPI_CS` until it becomes 1. Write character to register `SPI_FIFO`. Read and ignore character from `SPI_FIFO`.
      iii. Poll `DONE` of `SPI_CS` register until it becomes 1.

iv. Clear `TA` bit of `SPI_CS` register. This will pull chip select high.

You can test your SPI implementation by using procedures `MAX7219.TestMode` and `MAX7219.NormalMode` described in the following part of this assignment.

2 MAX 7219 Driver

The MAX7219 chip receives commands as 16-bit words on the SPI bus in the following way:

- The SPI driver pulls the select line low.
- The SPI driver transfers 16 bits that flow into the shift register of the MAX7219 chip. The MAX7219 chip shifts out the bits at the other end of the shift register to its data-out pin. The latter can be used in order to daisy-chain MAX7219 chips (how?).
- The SPI driver pulls the select line high. On the rising edge of the select line, the 16 recently transferred bits are latched and interpreted as commands by the 7219 chip.

The commands of the chip are described from page 7 of the MAX7219 manual. They are referred to as registers. Tasks:

1. In order to test if your SPI driver works at least partially, use the simple commands `0x0f00` and `0x0f01` to switch off and on the display test mode, respectively. This has already been implemented with procedures `MAX7219.TestMode` and `MAX7219.NormalMode`.
2. Implement a function to display a given 32-bit integer value as a hex-number on the eight 7-segment characters of the display. Constants containing the bit-pattern representing numbers 0 to 9 and characters a to f are provided in the skeleton of this exercise.
3. Optional: display the (hex)value of the kernel timer value (procedure `Kernel.GetOSTimer()`) on the 8-digit display using a 20ms periodic task.

Finally, if you cannot get enough SPI then implement a bit-banging version of the SPI driver (i.e. program the pins yourself bypassing the SPI controller)!

Documents

- System Construction Lecture 5 slides from the course-homepage `http://lec.inf.ethz.ch/syscon`
- BCM2835 manual
- Manual of the MAX7219 8-digits 7-segment display chip.

Some Code Patterns

- Important constants for this exercise:
  `Platform.GPFSEL0`, `Platform.GPFSEL1` (GPIO select registers)
  `Platform.SPI_CLK`, `Platform.SPI_CS`, `Platform.SPI_FIFO` (SPI0 registers)
- **Convert an integer number** `i` **into character** `c`**:** `c := CHR(i)`
- **Convert a character** `c` **into an integer** `i`**:** `i := ORD(c)`
- **Include a number in a set** `INCL(s,i)`
- **Remove a number from a set** `EXCL(s,i)`
- **If a number** `i` **is contained in a set** `s`**:** `i IN s`
- **Write an integer** `i` to a memory mapped register `r`: $[r] \leftarrow i$
  `Platform.WriteWord(r, i)`
- **Write a (bit) set** `s` to a memory mapped register `r`: $[r] \leftarrow s$
  `Platform.WriteBits(r, s)`
- **Set bits** `s` of a memory mapped register `r`, leaving all other bits unchanged: $[r] \leftarrow [r] \cup s$
  `Platform.SetBits(r, s)`
- **Clear bits** `s` of a memory mapped register `r`, leaving all other bits unchanged: $[r] \leftarrow [r] \cap \neg s$
  `Platform.ClearBits(r, s)`
- **Clear and set bits** `s` in a mask `m`, leaving all other bits unchanged: $[r] \leftarrow [r] \cap \neg m \cup s$
  `Platform.ClearAndSetBits(r, m, s)`