ELEC3850 - Embedded Systems 1

Real World Development

Brenton Schulz

University of Newcastle

August 19, 2020

Summary

- Sample development timeline
  - Pick a project requirement
  - Identify hardware module which meets requirement
  - Read datasheets - identify microcontroller interface
  - Work out how to drive microcontroller interface with your system
    - Bit-banging software interface?
    - Existing microcontroller peripheral (I2C, SPI, etc)?
    - Hardware configuration?
    - Do you need to write a driver library?
    - Are there complex interrupt / DMA timing constraints?

Pick An Imaginary Project Requirement

- Display information to the user with an 8x8 pixel LED matrix

Pick An Imaginary Project Requirement

- Display information to the user with an 8x8 pixel LED matrix
- Lets search eBay for parts...

Pick An Imaginary Project Requirement

- Display information to the user with an 8x8 pixel LED matrix
- Lets search eBay for parts...
- Great, we pick the MAX7219
- Why? Is this good design?
  - It meets requirements
  - It is available
  - (Presumably) it is within budget

MAX2719

- Note: There are other factors we have not considered
  - Update rate
  - Size
  - Power requirements
  - Heat output
  - ...etc
- It might not make it to the final design!
- Be preapred to re-evaluate after prototyping.

MAX7219

- Lets read the datasheet
  - How do we find it?
  - What information are we looking for?

MAX7219

- Lets read the datasheet
  - How do we find it?
  - What information are we looking for?
    - Digital interface specifications
  - In practice: flick over everything as there might be surprises

MAX7219

- Lets read the datasheet
  - How do we find it?
  - What information are we looking for?
    - Digital interface specifications
  - In practice: flick over everything as there might be surprises
- Note that (annoyingly) the timing diagram is on page 6 but the timing data is back on page 3
- What other crucial information does the datasheet have?

MAX7219

- Lets read the datasheet
  - How do we find it?
  - What information are we looking for?
    - Digital interface specifications
  - In practice: flick over everything as there might be surprises
- Note that (annoyingly) the timing diagram is on page 6 but the timing data is back on page 3
- What other crucial information does the datasheet have?
  - Serial data format
  - Register map

MAX7219

- Ok, we have data. What now?

MAX7219

- Ok, we have data. What now?
- Observe the register map - what needs to happen before we display data?

MAX7219

- Ok, we have data. What now?
- Observe the register map - what needs to happen before we display data?
- Device initialisation!
  - Power-on settings might (or might not) be random
  - Initialisation sets the configuration registers to a known or wanted state
  - This is needed for almost all peripheral modules

MAX7219 - Initialisation

- Pen and paper time
- Re-read datasheet
- Note all configuration registers and calculate initial values for your application

MAX7219 - Initialisation

- To save time, have a summary:
  - NB: `X` indicates "don't care" in the address data.
  - Table below assumes these bits are zero for clarity

Register	Address	Set Value	Notes
Digit Data	0x01	Don't care	Data will be filled in later
Decode Mode	0x09	0x00	No decode - driving matrix
Intensity	0x0A	0x0F	Full brightness
Scan Limit	0x0B	0x07	Drive all pixels
Shutdown	0x0C	0x00	Normal Operation
Display Test	0x0F	??	0x01 to test, 0x00 otherwise

- Does the initilisation order matter? Does shutdown have to be turned off first? Sometimes this matters, sometimes it doesn't.

MAX7219 - Software

- Programming time - What to do?
- Brainstorm some useful functions:
  - `MAX7219_WriteData();`
  - `MAX7219_Init();`
  - `MAX7219_DrawAll();`
  - `MAX7219_WriteRow();`
- This is otherwise known as writing a *driver*

Bit Banging

- Slang term - see Wikipedia for more details
  `https://en.wikipedia.org/wiki/Bit_banging`
- Can refer to many bit manipulations
- Most commonly refers to a software-driven serial data interface
- Compromises:
  - Useful when hardware interface is not available
  - Can be debugged at low speed using instruction stepping
  - A good exercise for students
  - Much slower and resource intensive than using dedicated hardware

MAX7219_WriteData()

- Implementation options:
  - Bit bang'ed
  - SPI peripheral
    - Polled
    - Interrrupt driven
    - DMA'ed

MAX7219_WriteData()

- Implementation options:
  - Bit bang'ed
  - SPI peripheral
    - Polled
    - Interrrupt driven
    - DMA'ed
- Start with bit banging
  - Sometimes you need it
  - It isn't taught anywhere
  - Low-level bit manipulation is tricky but sometimes necessary
    - eg: WS2812 RGB LEDs and DS18B20 temperature sensors use proprietary protocols
- Use SPI later - once everything else it working

Bit Banging

- Example, assuming a synchronous (clocked) serial interface
- General algorithm:

```
BEGIN
  uint8_t data;

  FOR each bit in `data`
    Place bit on an output pin
    Cycle a clock pin
  ENDFOR
END
```

- Typically needs >> or << shift operators and bitwise logic operators

Bit Banging

- With more detail, assuming MSB first:

```
uint8_t data, i;

for(i = 0; i < 8; i++)
{
  OUTPUT_PIN = (data & 0x80) >> 8;
  data = data << 1;
  CLK_PIN = 1;
  CLK_PIN = 0;
}
```

- Assumes data is latched on the *rising* edge
- `(data & 0x80)` is either zero or 128 (`0x80`). Shift right by 8 bits to make this zero or 1.

MAX7219 Interface

- Check the MAX7219 datasheet again
- We need to know two things:
  - The state the clock idles at
  - The clock edge which latches data
- Make a note of these facts somewhere

STM32CubeIDE

- Next step: STM32 configuration
- Initial coding will be done with bit banging
- Later we plan to use SPI
- Therefore: choose pins that also connect to SPI
  - CubeMX will tell you which pins these are
  - Check dev board datasheets to work out where they are physically
  - Configure them as GPIO outputs for now

STM32CubeIDE

- What else needs checking or modifying?

STM32CubeIDE

- What else needs checking or modifying?
- Probably clocks - by default the CPU clock may not be optimal
  - Note: Maximum clock isn't always best!
  - There is a small power/heat compromise when increasing clock rate

STM32CubeIDE

- What else needs checking or modifying?
- Probably clocks - by default the CPU clock may not be optimal
  - Note: Maximum clock isn't always best!
  - There is a small power/heat compromise when increasing clock rate
- Generate the project
- Observe HAL files which were included

STM32CubeIDE - HAL

- How do we learn about the HAL?
- Documentation for STM32F4: `https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal.pdf`
- HAL requires solid C programming knowledge
  - Lots of pointers...everywhere
  - Lots of structures

STM32CubeIDE - HAL

- Aside: why does it use so many structures?

STM32CubeIDE - HAL

- Aside: why does it use so many structures?
- eg:

```
1  typedef struct
2  {
3    __IO uint32_t CR1;
4    __IO uint32_t CR2;
5    __IO uint32_t SR;
6    __IO uint32_t DR;
7    __IO uint32_t CRCPR;
8    __IO uint32_t RXCRCR;
9    __IO uint32_t TXCRCR;
10   __IO uint32_t I2SCFGR;
11 } SPI_TypeDef;
```

STM32CubeIDE - HAL

- Check the SPI memory map in the reference manual: `https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405415-stm32f407417-pdf`
- This is on Page 916
- Compare register map to the `SPI_TypeDef` structure...

STM32CubeIDE - HAL

- Why is this structure:register mapping useful?

STM32CubeIDE - HAL

- Why is this structure:register mapping useful?
- Structure pointers!
- A `SPI_TypeDef*` pointer can access a register with `->`
- eg:

```
1 SPI_TypeDef *spi1 = 0x40013000; // SPI1 base
    address
2 spi1->DR = dataByte; // Write dataByte to tx reg
```

- The SPI1 base address is built into the HAL - no explicit need to do things this way

STM32CubeIDE - writing `MAX7219_WriteData()`

- Finally some coding!
- Lets write the low-level function
- Then we will *test* the function
  - ALWAYS TEST!
  - Use a logic analyser - Saleae units are in the lab
  - Have a hand-written output to compare against

STM32CubeIDE - writing `MAX7219_WriteData()`

- Finally some coding!
- Lets write the low-level function
- Then we will *test* the function
  - ALWAYS TEST!
  - Use a logic analyser - Saleae units are in the lab
  - Have a hand-written output to compare against
- Next step: write the initialisation
- Test init works with the lamp test register
- Write the other functions after

MAX7219 - converting to SPI

- Are the other functions working now? Can we draw to the LED matrix?
- Next step: replace bit-banging with SPI
- Development steps:
  - Re-configure device for SPI in CubeMX
  - Replace bit-banging code with a call to HAL_SPI_Transmit();
  - NB: the `CS` line needs to be manually controlled if cascading multiple modules
  - Try to use the HAL files as documentation
    - What is in the .h files?
    - What is in the .c files?
    - Make use of "Open Declaration" feature