CSD example problems and projects

Chapter 3: microcontroller applications
Collection of former problems under revision for adaptation to CSD new *digsys* format.
3.1. The microcontroller PIC16F

Answer the following questions referred to the microprocessors in general:

1. Explain the differences between Harvard (Microchip PIC) and Von Neumann (Intel 8051) microprocessor architectures. Draw the sketch of the architectures.
2. Which is the main architectural difference between 8/16/32 bits microprocessors?
3. Which are the functions of the FLASH (ROM) memory and the RAM registers?
4. Explain what the stack memory is and how it is used for.
5. Describe the main blocks of the central process unit (CPU) and how can you connect it to the content of the previous Chapters 1 and 2.
6. How an assembler instruction is executed? Find an example of C code disassembled and explain how it works.
7. How many clock cycles are required for executing an instruction in assembler?

The architecture of the PIC16F87xA family is presented in Fig. 1.
- Examine it and list the components that you could be able to design and synthesise, if that were the case, into a PLD using VHDL and applying strategies from previous chapters.
- Can you redraw the architecture as a programmable dedicated processor to solving each machine-code instruction?
- Draw the blocks of the RAM and the ROM components and explain how to perform memory writing and reading operations.
- Find the specification of the Timer0 peripheral and compare them with the programmable timer designed in Chapter 2.
Fig. 1
PIC16F87xA architecture.

<table>
<thead>
<tr>
<th>Device</th>
<th>Program Flash</th>
<th>Data Memory</th>
<th>Data EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC16F874A</td>
<td>4K words</td>
<td>192 Bytes</td>
<td>128 Bytes</td>
</tr>
<tr>
<td>PIC16F877A</td>
<td>6K words</td>
<td>356 Bytes</td>
<td>256 Bytes</td>
</tr>
<tr>
<td>Mnemonic, Operands</td>
<td>Description</td>
<td>Cycles</td>
<td>14-Bit Opcode</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>--------</td>
<td>---------------</td>
</tr>
<tr>
<td><strong>BYTE-ORIENTED FILE REGISTER OPERATIONS</strong></td>
<td></td>
<td></td>
<td>MSb</td>
</tr>
<tr>
<td>ADDWF f, d</td>
<td>Add W and f</td>
<td>1</td>
<td>00 0111 dfff ffff</td>
</tr>
<tr>
<td>ANDWF f, d</td>
<td>AND W with f</td>
<td>1</td>
<td>00 0101 dfff ffff</td>
</tr>
<tr>
<td>CLRF f</td>
<td>Clear f</td>
<td>1</td>
<td>00 0001 lfff ffff</td>
</tr>
<tr>
<td>CLR f</td>
<td>Clear W</td>
<td>1</td>
<td>00 0001 0xxx xxxx</td>
</tr>
<tr>
<td>COMF f, d</td>
<td>Complement f</td>
<td>1</td>
<td>00 1001 dfff ffff</td>
</tr>
<tr>
<td>DEC f, d</td>
<td>Decrement f</td>
<td>1</td>
<td>00 0011 dfff ffff</td>
</tr>
<tr>
<td>DECFSZ f, d</td>
<td>Decrement f, Skip if 0</td>
<td>1 (2)</td>
<td>00 1011 dfff ffff</td>
</tr>
<tr>
<td>INC f, d</td>
<td>Increment f</td>
<td>1</td>
<td>00 1010 dfff ffff</td>
</tr>
<tr>
<td>INCFSZ f, d</td>
<td>Increment f, Skip if 0</td>
<td>1 (2)</td>
<td>00 1111 dfff ffff</td>
</tr>
<tr>
<td>IORWF f, d</td>
<td>Inclusive OR W with f</td>
<td>1</td>
<td>00 0100 dfff ffff</td>
</tr>
<tr>
<td>MOVW f</td>
<td>Move W to f</td>
<td>1</td>
<td>00 1000 dfff ffff</td>
</tr>
<tr>
<td>MOVWF f</td>
<td>Move f to W</td>
<td>1</td>
<td>00 1000 0fff ffff</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>00 0000 0000 0000</td>
</tr>
<tr>
<td>RLF f, d</td>
<td>Rotate Left f through Carry</td>
<td>1</td>
<td>00 1101 dfff ffff</td>
</tr>
<tr>
<td>RRF f, d</td>
<td>Rotate Right f through Carry</td>
<td>1</td>
<td>00 1110 dfff ffff</td>
</tr>
<tr>
<td>SUBWF f, d</td>
<td>Subtract W from f</td>
<td>1</td>
<td>00 0010 dfff ffff</td>
</tr>
<tr>
<td>SWAPF f, d</td>
<td>Swap nibbles in f</td>
<td>1</td>
<td>00 1110 dfff ffff</td>
</tr>
<tr>
<td>XORWF f, d</td>
<td>Exclusive OR W with f</td>
<td>1</td>
<td>00 0110 dfff ffff</td>
</tr>
<tr>
<td><strong>BIT-ORIENTED FILE REGISTER OPERATIONS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCF f, b</td>
<td>Bit Clear f</td>
<td>1</td>
<td>01 00bb bfff ffff</td>
</tr>
<tr>
<td>BSF f, b</td>
<td>Bit Set f</td>
<td>1</td>
<td>01 01bb bfff ffff</td>
</tr>
<tr>
<td>BTFS f, b</td>
<td>Bit Test f, Skip if Clear</td>
<td>1 (2)</td>
<td>01 10bb bfff ffff</td>
</tr>
<tr>
<td>BTFFS f, b</td>
<td>Bit Test f, Skip if Set</td>
<td>1 (2)</td>
<td>01 11bb bfff ffff</td>
</tr>
<tr>
<td><strong>LITERAL AND CONTROL OPERATIONS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDLW k</td>
<td>Add Literal and W</td>
<td>1</td>
<td>11 111x kkkk kkkk</td>
</tr>
<tr>
<td>ANDLW k</td>
<td>AND Literal with W</td>
<td>1</td>
<td>11 1001 kkkk kkkk</td>
</tr>
<tr>
<td>CALL k</td>
<td>Call Subroutine</td>
<td>2</td>
<td>10 0kkk kkkk kkkk</td>
</tr>
<tr>
<td>CLRWDT -</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>00 0000 0110 0100</td>
</tr>
<tr>
<td>GOTO k</td>
<td>Go to Address</td>
<td>2</td>
<td>10 1kkk kkkk kkkk</td>
</tr>
<tr>
<td>IORLW k</td>
<td>Inclusive OR Literal with W</td>
<td>1</td>
<td>11 1000 kkkk kkkk</td>
</tr>
<tr>
<td>MOVLW k</td>
<td>Move Literal to W</td>
<td>1</td>
<td>11 00xx kkkk kkkk</td>
</tr>
<tr>
<td>RETFIE -</td>
<td>Return from Interrupt</td>
<td>2</td>
<td>00 0000 0000 1001</td>
</tr>
<tr>
<td>RETLW k</td>
<td>Return with Literal in W</td>
<td>2</td>
<td>11 01xx kkkk kkkk</td>
</tr>
<tr>
<td>RETURN -</td>
<td>Return from Subroutine</td>
<td>2</td>
<td>00 0000 1000 1000</td>
</tr>
<tr>
<td>SLEEP -</td>
<td>Go into Standby mode</td>
<td>1</td>
<td>00 0000 0110 0011</td>
</tr>
<tr>
<td>SUBWF k</td>
<td>Subtract W from Literal</td>
<td>1</td>
<td>11 110x kkkk kkkk</td>
</tr>
<tr>
<td>XORLW k</td>
<td>Exclusive OR Literal with W</td>
<td>1</td>
<td>11 1010 kkkk kkkk</td>
</tr>
</tbody>
</table>
3.2. Invent a Dual_MUX4 based on a µC

This assignment and tutorial can be found here.

3.3. 1-digit BCD adder

The specifications of this project are simply add two 1-digit BCD numbers considering as well the \( C_{in} \) and the \( C_{out} \) signals to be able to chain components of the same kind. The circuit to solve is represented in Fig. 3.

Remember that, as usual, you have to organise the documentation to hand in in 4 sections, each one in different sheets of paper. Section 5, prototyping, is always optional in case you like to invest some more time in the laboratory downloading the microcontroller configuration program (hex) to the training board while measuring and characterising the prototype using workbench instrumentation.

The problem is reviewed and assessed in this way: half of the project, sections 1 and 2, is prepared in classrooms using paper and group discussions. The last sections 3 and 4 are solved by means of the virtual laboratory (IDE – Simulator) available from our virtual desktop computers. Remember that each of you have access to your personal network disk (L:) to properly develop and test your own project.

- Specifications and planning → 5p.
- Development and test → 5p.

A note on group discussions. Learning to work cooperatively is not an easy task, but indeed, very demanding. It doesn’t mean that one of you has to do
all the work and the others simply have to copy and paste. On the contrary, use cooperation and group dynamics to clarify your ideas and to organise the project. Thus hand in an individual report (you are the responsible and the only author of all the material):

- Handmade specifications, tables, symbols, diagrams, timing diagrams, bullet list, etc.
- Handmade schematics, block diagrams, annotations, flowcharts, comments, etc.
- Write your own code and draw your own schematics. Print and comment schematics and C code. This is how you can print correctly your C files.
- Print the project compilation results to shown how your IDE has generated no errors. Annotate how many RAM memory bytes has been used and discuss whether there is an agreement with your initial planning of variables. Explain how long is your Flash program. Explain the difference between the COF and the HEX configuration files. Print a portion of disassembled code and explain details on how the data is transferred or the operations are executed. Etc.
- Print test results with meaningful annotations and discussion on the results to demonstrate that the project works as expected. Measure how long does it take to run the main loop code. Measure the time required to execute a single assembly instruction.

And on top of that, be fair: you are here at this university to learn the content of this subject deeply, not superficially. A hardware engineer has to be able to project circuits professionally and successfully, and to meet this goal, it is required a complete personal involvement and engagement. You can count as well with our commitment to help you any time.

A tutorial on the project can be read here.
3.4. 12-to-4 encoder
The project objective is to design an encoder $Enc_{12\_4}$ like similar to the one presented in problem Error! Reference source not found. using a µC. For instance, the application can be integrated as a subsystem in a professional PBX door phone with dialling keypad as represented in Fig. 4. The device has to generate the 4-bit code of the clicked key. BCD codes for keys from 0 to 9, and “1010” for the hash symbol “#” and the “1011” symbol for the asterisk symbol “*”. The group select output (GS) has to be held high when any key is pressed. The encoder also has an enable input (Ei) and an enable output Eo to detect when the encoder is disabled or when is active but no one is pressing keys. Finally, a 7-segment output will represent the code of the key pressed.

The emphasis is set therefore in learning basic polling digital inputs and writing digital outputs. The truth table will be solved using a behavioural plan B interpretation in C language, organising the hardware-software diagram so that the input and outputs variables will be held in RAM memory. Your planning has to be similar to the one discussed in Chapter 1, but using C code instead of VHDL. The symbol for the $Enc_{12\_4}$ that will have priority high as usual in this kind of devices is represented in Fig. 5.
1. **Specifications.** Draw the truth table for this $Enc_{12\_4}$.
2. **Plan.** Propose a hardware-software diagram naming all the electrical signals, RAM variables and the software functions.
**Hardware.** Copy and adapt a circuit from any of the previous projects and name it *Enc_12_4.pdsprj*. Make the pin assignment connections accordingly to the options *a*, *b* or *c* given by your instructors.

**Software.** Explain how to configure the μC in *init_system()*. Organise using a flowchart the interface function *read_inputs()*. Organise using a flowchart the interface function *write_outputs()*. Infer the *truth_table()* software function using a behavioural interpretation and the corresponding flowchart.

![Symbol of the Enc_12_4 and the idea of designing it using a μC.](image)

You can solve in a phase#1 the circuit with GS, EI, Eo, and the BCD outputs Y, and later in a phase#2 you can add the feature of the 7-segment interface including the other S_L signals.

3. **Developing.** Write the *Enc_12_4.c* source code translating the function flowcharts. Start a software IDE project for the target microcontroller PIC18F4520 and generate the configuration files “.cof” and “.hex” after compilation. Discuss the project summary: % of ROM used for the code, number of RAM bytes used, etc. Find the RAM memory position of the variable *Var_D*. How many bytes does it occupy?

4. **Test.** Do it interactively in Proteus every time a few lines of code are added to the source file. Measure how long does it take to run the main loop code when using a 4 MHz oscillator.
3.5. 1-digit BCD counter

This is a plan X example to relate the design of FSM to Chapter 2.

+ A tutorial on this project is available [here](#).

3.6. Binary counter modulo 256

This is a plan Y example to relate the design of FSM to Chapter 2.

+ Project files are available [here](#).

3.7. 4-bit serial data transmitter

Let us design a simple 2-wire asynchronous data transmitter based on a μC for sending to another computer a nibble (4-bits) of data. It is basically a right-shift register. The application symbol and pinning of the PIC18F4520 is represented in Fig. 6. We’ll use the FSM style of programming in C language. The format for the serial output once the start-transmission ST rising edge is detected by means of an interrupt is: Start-bit (‘0’), Data_in (0), Data_in (1), Data_in (2), Data_in (3); and then the end-of-transmission EoT pulse is generated to indicate that the transmitter has ended the process (see the Fig. 7). Serial_out is held high when idle.
a) Draw the hardware schematic. Reset circuit, XTAL oscillator, *Data_in(3..0)* = (RA2, RA1, RD7, RD6), *CLK* (RB0), *ST* (RB1), *Serial_out* (RC5), *EoT* (RC2). Explain how to configure the inputs and outputs in the *init_system()*.

---

**Fig. 6**
Symbol of the data transmitter and the µC PIC18F4520 from Microchip.

**Fig. 7**
Example of a section of a timing diagram where it can be seen how the data is read and right shifted in a single wire.
b) Draw the hardware/software diagram indicating the required RAM variables and how the FSM is solved in software. The transmission sequence will start when a rising edge is detected at the start ST push button by means of the interrupt INT1IF. The CLK input will generate an interrupt INTOIF so that a new bit is transmitted at a time at the Serial_out as represented in Fig. 7. Transmission speed is 150 b/s.

c) How read_input() works to generate the char variable var_Data_in?

d) How the variables var_Serial_out and var_EoT are written to the corresponding pins using write_outputs() without interfering the other µC port pins?

e) Which is the ISR() used for? Propose the flow chart.

f) Draw an state diagram showing the state transitions and the outputs for each state. Name the states, for instance: Idle, Start_bit, Data_0, Data_1, etc.

g) Draw the truth tables and their equivalent flow charts for the state_logic() and output_logic().

h) How to use and program the TMR0 peripheral in 8-bit mode to replace completely the functionality of the external CLK as the baud-rate generator?

A tutorial on the project can be read here.

3.8. 5-bit Johnson counter

A tutorial on this project is available in these references (1), (2).
3.9. Stepper motor controller

Design the digital control unit (stepper_controller) for the “9904 112 31004” stepping motor from Premotec shown in Fig. 8 following our microcontroller-based strategy. Today stepper motors can be found in computer peripherals, machine tools, medical equipment, automotive devices, or small business machines, to name a few applications. Clockwise (CW) and counter-clockwise (CCW) rotation can be achieved by reversing the step sequence. Inhibit (INH) is like a count disable, do not letting the motor rotate. Step or stride angle is 7.5 degree, thus 48 CLK periods are required for a full revolution. External CLK frequency is 96 Hz, and so when running it rotates at 2 revolutions per second (Fig. 12). The idea is to connect four port pins to the motor coils and drive them with the right sequence so that the motor inhibits or rotates clockwise or counter-clockwise accordingly to the input signals INH and CW. Four additional pins are used connected to LED to visualise the coils binary sequence.

Clockwise (‘1’) - Counter-clockwise (‘0’)

Fig. 8 Example of two-phase stepper motor: characteristics, connections and unipolar winding circuit using a power driver to energise coils.
1. Draw the schematic: input switches, outputs, reset (MCLR_L) and 4.8 MHz quartz crystal oscillator OSC.
2. Draw the hardware-software diagram. Why the rotation CLK block has to be connected to RB0/INT pin? What the interrupt service routine `ISR()` is used for?
3. Organise and name RAM variables for the project. Explain how to configure port pins and interrupts in `init_system()`.
4. Explain how to poll the input values using bitwise operations in `read_inputs()`.
5. Explain how to drive the eight outputs using bitwise operations in `write_outputs()`.
6. Draw the truth table and flowchart for the `output_logic()`.
7. Draw the truth table and flowchart for the `state_logic()`.
8. Replace the external CLK configuring the embedded TMR0 (Fig. 10 in 8-bit mode) to obtain the same 96 Hz step frequency.

**Problem discussion.**

---

**Fig. 9**
Full wave stepping sequence and unipolar winding circuit using a power driver to energise coils. Proposed state diagram to control the FSM.

**Fig. 10**
TMR0 schematic in 8-bit mode.
3.10. LCD display using ASCII messages and static data

+ A tutorial on this project is available in this reference (1).

3.11. LCD display using dynamic data

+ A tutorial on this project is available in this reference (1)

3.12. Interfacing an I2C display

+ A tutorial on this project is available in this reference (1)

3.13. Industrial application

This problem is connected with problem XX. The idea now is to design it using a microcontroller instead of a dedicated hardware design in VHDL.

Continue the problem as follows:

5. Draw the hardware schematic for an Atmel ATmega8535 microcontroller. Connect all the inputs and outputs to convenient I/O port pins, the reset button and the 12 MHz quartz crystal.

6. Architecture of the software. Organise and describe the program variables. Explain the use of interrupts. Assume that in a Phase #1 of the design, an external CLK signal of 4 Hz is available to generate the
warm, hot and cold-water timing periods of 50, 10 and 20 s respectively.

7. Describe the flowchart of bitwise operations for the functions to interface the hardware: \textit{read\_inputs()}, \textit{write\_outputs()} and \textit{ISR(interrupt service routine)}. What kind of operations are solved by the \textit{init\_system()} function?

8. Describe the truth table and flowchart of the function to solve the state transitions: \textit{state\_logic()}.

9. Describe the truth table and flowchart of the function to implement the output variables: \textit{output\_logic()}.

10. Explain how to implement the timing signal of 4 Hz s internally using the Timer0 counter/timer peripheral in a Phase #2 of the project.
3.14. **Simple remote control**

We want to design a very simple wireless infrared remote control for an electronic equipment as shown in Fig. 11. In this initial stage of the design, the Chip1 is the microcontroller while the other components are external integrated circuits. The Chip2 (decoder BCD to 7-segments) is used to show the channel number, the Chip3 is the infrared transmitter, and the Chip4 is a 2 seconds CLK. Furthermore, the volume control is not implemented and thus only the buttons **BU** and **BD** are considered.

The system has a capacity of 7 channels. To increment the channel the BU (Channel up) has to be pressed. To decrement the channel number the BD (Channel down) has to be pressed. If both buttons are pressed or released simultaneously, the channel count is maintained. The buttons are sampled every 2 seconds (0.5 Hz). The outputs C(2..0) represents the channel selected in binary.

![Schematic of a basic remote control for 7 channels.](image)

**Design phase #1**

1. Timing diagram.
2. State diagram. The initial state is the Channel 1.
3. Hardware circuit. Connect input and output pins to the microcontroller ATmega8535, a master reset and crystal oscillator of 8 MHz.
5. Functions to interface the hardware: `init_system()`, `read_inputs()`, `write_outputs()`, `ISR(source of interrupt)`. Flowchart of bitwise operations.
6. The function to implement state transitions: `state_logic()`. Truth table and flowchart.
7. The function to implement the outputs: `output_logic()`. Truth table and flowchart.

**Design phase #2**

8. If we like to include into the microcontroller the functionality of the Chip2 (decoder BCD to 7-segments), so that this external chip will be no longer required, how to proceed?
9. If we like to include the Power_ON/OFF button into the microcontroller, so that when clicked the code “000” is generated immediately, how to proceed?

3.15. Non-retriggerable timer

1. Specifications
Our aim is to design a timer of exactly 11.25 s as represented in Fig. 12. It is non-retriggerable, which means that the system is not affected even if you click the Trigger more than once while active in the timing period. In this project, the strategy will be to count external/internal pulses once the trigger signal is detected as represented in the timing diagram. The project is based on a PIC18F4520 microcontroller from Microchip. We will consider two design options and you have to choose one of them:

**Option A:** Using the external 16 Hz clock input as the INT1 interrupt source.

**Option B:** Using the internal TMR0 peripheral instead of the 16 Hz clock input.

![Circuit symbol and waveforms for the "Timer" project. In option B the CLK (INT1) is not connected because the internal Timer0 is used instead.](image)
2. **Planning.** A microcontroller-based architecture running a FSM.

1. **Hardware:** draw the schematic indicating where to connect and how the system oscillator, the reset circuit, and all the remaining inputs and outputs.

2. **Software:** Draw a possible state diagram for the timer system. How many states will this FSM contain? Which is the task to be performed in each state?

3. Infer all the software variables (names and types) that will be required for managing the application. How many bytes of RAM memory will require? What kind of variable is `current_state`?

4. Explain how to organise the software (main, setup, interrupts, write outputs, etc.) and describe the operations to setup the system `init_system()` and explain how to set a pin as input or output.

5. Describe the bitwise operations and the flow chart required to write the output: `write_outputs()`.

6. Write the C code lines of the interrupt service routine `ISR()`.

7. Describe the truth table and how to organise the flow chart of the `state_logic()` function.

8. Describe the truth table and how to organise the flow chart of the `output_logic()` function.

3. **Development and D. Verification**

9. Which EDA and debugging tools and techniques are you going to use to compile the code and test the system in Proteus?

10. **(extra)** Download the microcontroller’s configuration file to the PICDEM 2 Plus board and verify how it works connecting the LED at the `T_out` output. If the Option A was chosen, use the laboratory’s signal generator to obtain the 16 Hz square wave that has to be applied at pin RB1/INT1.
3.16. Timers. PWM generation

Fig. 13 shows the symbol of an application based on the PIC18F4520 (Fig. 14) running with an 8 MHz crystal quartz oscillator. The idea is to control the rotation speed of a direct current motor generating a 25 Hz waveform that has 2 possible selectable duty cycles: DC1 = 20% and DC2 = 80%. The 7-segment display will show the sign “.” when idle, and the numbers 1 or 2 depending on the DC selected by the switch. The button B starts and stops the waveform generation. Fig. 15 shows an idea of the state diagram.

Fig. 13
This is an idea of the state diagram proposed to run this application. It must be completed.

Fig. 14
Pinning of the microcontroller PIC18F4520

![Image of the application symbol and state diagram](image)

- a) Draw the two waveforms indicating the $T_{ON1}$, $T_{OFF1}$, $T_{ON2}$ and $T_{OFF2}$ periods of time.
- b) Draw the schematic connecting the inputs and outputs to the PIC18F4520. Add the crystal oscillator and the MCLR_L circuits. Explain how to configure the inputs and outputs in the `init_system()`.
- c) Explain how to connect and configure the TMR0 (Timer 0) peripheral to generate interrupts. Which are the necessary $N1$ and $N2$ values for the pre-scaler and the TMR0 counter to be able to generate all the required timing periods?

$Timing\_period = \frac{4}{F_{OSC}} \cdot N1 \cdot N2$
d) Draw the hardware/software diagram indicating the required RAM variables and how the FSM is solved in software. How to implement the functions `read_inputs()`, `write_outputs()` and `ISR()`? How and where to drive the 7-segment display to show the sign “-“ and the numbers “1” and “2”?

e) Complete the state diagram represented in Fig. 15 and deduce the truth tables for the main functions of the C code: `state_logic()` and `output_logic()`.

![State Diagram](image)

[Fig. 15]
This is an idea of the state diagram proposed to run this application. It must be completed.

+ Problem discussion.