# EEAP 282
# PROGRAMMING ASSIGNMENT #6

## EMBEDDED CONTROLLER

ABSTRACT: At various points throughout the course references have been made to the use of embedded microprocessors in appliances and other consumer products. In this lab you will implement a basic appliance controller for a deep fat fryer similar to those used at McDonald's and other fast food chains foor cooking french fries. Your program will control the cooking of french fries according to two criteria: (1) elapsed time as measured by an interrupt driven clock, and (2) the temperature of the cooking oil as determined by a digital temperature sensor. Part (1) is required to get out of the final, Part (2) is extra credit.

Technical details:

PART 1: You want to time an event using interrupts which are produced at periodic intervals. To do this you will need to program a 68230 interrupt/timer (PIT) chip to produce interrupts at the rate of 60 interrupts/second. The interrupts will start upon input of a control signal from the keyboard. Your program will count these interrupts and update a cooking display whenever a second of time (i.e. 60 interrupts) has elapsed. When the cooking time reaches zero the cooking of the product will be complete. Your program should accept an input cook time (minutes, seconds) and will consist of an interrupt service routine that counts interrupts and a main program that displays the current cooking time remaining. Note that this will require that you convert your data from hex to ASCII.

The 68230 programmed interface/timer (PI/T) chip is emulated in the debugger by appropriate macros. Note that the 68230's interrupt SHOULD be reset by clearing the ZDS bit in the interrupt service routine.

OPTIONAL: To make the program more interesting you can program it to beep when the cook cycle is done. The beeping sound can be generated by sending a $07 to your terminal.

The program should have the following programming sequence:

1. Request the input cook time
2. Store the input cook time
3. Initialize registers in the PI/T
4. Reset the 68230 status bits
5. Set up the interrupt vectors
6. Enable the timer
7. Enable interrupts by changing the status register
8. Generate displays and beeps as necessary

The terminal display function can be done using HexOut.

HINTS/NOTES ON DEBUGGING YOUR LAB

1. Put your display routine in your main program loop. Display timing is not critical and will slow down your interrupt service routine. DO NOT call exceptions (TRAP's) in your interrupt service routine.

2. IMPORTANT NOTE: You have to set an option in the debugger to properly process interrupts. At the command line type:

```
Debugger Options General Exceptions Normal
```

or

```
Debugger Options General Exceptions Report
```

to enable interrupts.

The default is

```
Debugger Options General Exceptions Stop
```

This command specifies what the debugger does on an exception. As one would guess Stop will simply display a message in the journal window and NOT process your exception. For debugging, I would recommend the Report option, it flashes a message in the journal window and processes the exception just as you would expect. When you actually run your clock program, use the Normal option which processes the exception the same way the Report option does BUT without the message to the journal window. This info is in the debugger manual on page 4-2.

3. Test a first version of your program without any fancy output. A major problem is often that your program is so complicated that you cannot debug it. Formatted output often causes problems.

4. You can examine the 68000 exception vector table using the debugger's Memory Display commands. You can also examine the 68230 registers. Note that the 68230 only has registers at odd addresses. Any attempt to display an even address (such as $10022) will generate a BUS ERROR.

PART 2: If you simply cook a product according to a predetermined time you will have large variation in the product due to variations in the actual temperature of the cooking oil. This can be compensated for by measuring the oil temperature and adjusting the cooking time accordingly. In a very common sense manner we can justify this by recognizing that if the oil temperature is below the nominal temperature the product will take longer to cook. If the oil temperature is above the nominal temperature the product will take less time to cook. An analog-to-digital (A/D) converter can take an input from the real world (a temperature signal in this case) and converts it into digital data that can be processed by the computer. We have provided a simulated A/D converter which must be communicated with using special memory locations called interface registers in a polled manner similar to the 6850 ACIA registers you used

in Lab #1. The programming interface for the A/D converter is described in Appendix 3.

What you should do is sample the oil temperature everytime you update the display (i.e. decrease the displayed cooking time by one second) and determine how many interrupts will actually determine the when you next update the displayed seconds. This can be done by counting the number of interrupts in a register and only updating the display when you have reached the desired count corresponding to the measured temperature. You will need to maintain separate registers which count the periodic interrupts and displayed cooking time. The relationship between these times is shown below.

| Temperature | ΔTemperature | Compensation |
|---|---|---|
| 300 | -50 | 89 |
| 305 | -45 | 84 |
| 310 | -40 | 79 |
| 315 | -35 | 74 |
| 320 | -30 | 71 |
| 325 | -25 | 67 |
| 330 | -20 | 65 |
| 335 | -15 | 63 |
| 340 | -10 | 61 |
| 345 | -5 | 60 |
| 350 | 0 | 60 |
| 355 | 5 | 60 |
| 360 | 10 | 59 |
| 365 | 15 | 57 |
| 370 | 20 | 55 |
| 375 | 25 | 53 |
| 380 | 30 | 49 |
| 385 | 35 | 46 |
| 390 | 40 | 41 |
| 395 | 45 | 36 |
| 400 | 50 | 31 |

Table 1. Number of cooking time interrupts as a function of temperature.

If the A/D converter indicates a temperature of 350 degrees, then 60 interrupts will determine one second of displayed time, i.e. the cooking time equals the real elapsed time. If the actual temperature increases to 360 degrees then the displayed cooking time will decrement one second for every 59 interrupts counted, if the temperature decreases to 330 degrees then the cooking time will decrement one second for each 65 interrupts counted, etc. The actual formula used to generate Table 1 was

$$COUNT = \begin{cases} 60 + 4 \times (T - 350)^2, T < 350°F \\ 60 - 4 \times (T - 350)^2, T \geq 350°F \end{cases}$$

What you need to do to implement this Part of your program is to modify your display algorithm from Part 1 so that when you update the cooking time display, you read the current temperature of the cooking oil and determine the number of interrupts to be counted before you NEXT update the cooking time display. This number of interrupts can be calculated according to Equation (1) or looked up in a table similar to Table 1 above. What is happening is that you are still measuring the actual time at at rate of 1/60-th of a second per interrupt, but the cooking time, i.e. rate at which the display will update, is a function of the temperature of the cooking oil.

To test your program we will provide simulated data which your A/D converted will read. Your program should keep track of the total number of interrupts which occur for a specified cooking time for this temperature profiles. Your grade for this part of the lab will be based strongly upon how accurately you determine the actual cooking time.

References:
This is the method actually used to cook french fries at most fast food chains throughout the world. Furthermore, the same approach can be used to time industrial processes such as the curing of automobile tires and other "rubber" products. What you are actually doing is solving the Arrehnius equation for a chemical reaction. It just happens that in this case the chemical reaction is the cooking of french fries.