

**Periodic Interrupts**

Data acquisition (Lab 7) samples ADC

Signal generation output to DAC

Audio player (Lab 8)

Communications

Digital controller

FSM

Linear control system (EE362K)

**Read Book Sections 9.1, 9.2, 9.4, 9.6.1, 9.6.2, 9.10**

Moore example in Metrowerks

Open example and run with FollowPC mode

1) Where does the 9S12 spend most of its time?

2) How do we recover this lost productivity?

```

const struct State{
    unsigned char Out;    // Output to Port T
    unsigned short Time; // Time in msec to wait
    const struct State *Next[4]; // if input =0,1,2,3
};
typedef const struct State StateType;
typedef StateType * StatePtr;

#define SA &fsm[0]
#define SB &fsm[1]
#define SC &fsm[2]
#define SD &fsm[3]
#define SE &fsm[4]
#define SF &fsm[5]
StateType fsm[6]={
    {0x01,2,{SB,SC,SD,SE}}, // SA,SB fast alternate toggle
    {0x02,2,{SA,SC,SD,SE}}, // SB
    {0x03,1,{SA,SC,SD,SE}}, // SC both on
    {0x00,1,{SA,SC,SD,SE}}, // SD both off
    {0x00,10,{SA,SC,SD,SF}}, // SE,SF together toggle
    {0x03,10,{SA,SC,SD,SE}} // SF
};

StatePtr Pt; // Current State
unsigned char Input;

//-----FSMInit-----
// initialize FSM, clock, initial state SA
// inputs: none
// outputs: none
void FSMInit(void){
    Pt = SA; // Initial State
    DDRT |= 0x03; // PT1,PT0 outputs
    DDRH &= ~0x03; // PH1,PH0 inputs
}

void main(void) {
    PLL_Init(); // 24 MHz
    Timer_Init(); // TCNT at 1.5 MHz
    FSMInit();
    asm cli
    for(;;) {
        PTT = (PTT&0xFC)+Pt->Out; // Output depends on state
        Timer_Wait1ms(Pt->Time); // Time to wait in this state
        Input = PTH&0x03; // Input=0,1,2,or 3
        Pt = Pt->Next[Input]; // Next state depends on input
    }
}

```

**Redesign using output compare interrupts**

**Foreground Solution**

- ; 1. Perform output for the current state
- ; 2. Wait for specified amount of time
- ; 3. Input from the switches
- ; 4. Go to the next state depending on the input
- ; 1. Perform output for the current state
- ; 2. Wait for specified amount of time
- ; 3. Input from the switches
- ; 4. Go to the next state depending on the input

...

*What is the computer doing most of the time?*

**Background Solution**

**Ritual**

- ; 1. Perform output for the current state
  - ; 2. Set TC0 to Wait for specified amount of time
- Output compare interrupt service routine
- ; 3. Input from the switches
  - ; 4. Go to the next state depending on the input
  - ; 1. Perform output for the current state
  - ; 2. Set TC0 to Wait for specified amount of time

Show Moore\_DP512 example

Move to background

More accurate

Frees up cycles to perform other tasks

Merge OC\_DP512 and Moore files

Debugging techniques when using interrupts

**Profiling**

- 1) Is the interrupt occurring? Is a function being called?
  - Add global counters, initialize to 0
  - Add **counter++** inside ISR, inside function
- 2) Is the interrupt occurring? Is a function being called?
  - Find unused I/O pins, initialize to outputs
  - Add **PTP |= 0x01;** at beginning of ISR, function
  - Add **PTP &= ~0x01;** at end of ISR, function
  - View bits with a logic analyzer or scope

Figure 9.14. A logic analyzer and example output.

**Pulse-width modulation (a way to adjust output power)**

Any diode is fine (1N914, 1N4001, 1N4002, 1N4003)

Motor interface using a high current MOSFET.

**Output compare every 1ms**

**Length** is a variable from 0 to 10

Every 10 interrupts make PM0 high

Every **Length** interrupts make PM1 low

Duty cycle is **Length**/10

Maximum power is  $V_m^2/R$

Delivered power is  $V_m^2/R * \text{Length}/10$

Build the output compare solution in Metrowerks

Simulate in TExaS

Run on actual board

