

Jonathan W. Valvano March 1, 2000, 11:00am-11:50am

**(45) Question 1.** In the first implementation, you will use a simple circular linked list and key wakeup interrupts.

(10) Part a) Show all global data structures required. Similar to Program 13.2

```
const struct State {
    unsigned char Out;           // Output to Port J
    const struct State *Next;}; // Next state
typedef const struct State StateType;
#define S5 &fsm[0]
#define S6 &fsm[1]
#define S10 &fsm[2]
#define S9 &fsm[3]
StateType fsm[4]={
    {0x05, S6},
    {0x06, S10},
    {0x0A, S9},
    {0x09, S5};
StatePtr *Pt; // Current State
```

(15) Part b) Show the ritual.

```
void Ritual(void){
    asm(" sei ");
    DDRJ=0x7F; // Unused pins are output
    KPOLJ=0x00; // fall on PJ7
    KWIEJ=0x80; // arm PJ7
    KWIFJ=0x80; // clear flag7
    Pt=S5; // initial state
    PORTJ=0x05;
    asm(" cli ");}
```

(10) Part c) Show the PORTJ key wakeup ISR, which should be executed on the fall of PJ7.

```
#pragma interrupt_handler KeyWakeupHan()
void KeyWakeupHan(void){
    KWIFJ=0x80; // clear flag, ack
    PORTJ=Pt->Out; // output for this state
    Pt=Pt->Next;} // Move to next state
```

(5) Part d) Show the C code that establishes the key wakeup interrupt vector.

```
#pragma abs_address: ffd0
void (*KeyWakeup_vector[])() = { KeyWakeupHan};
#pragma end_abs_address
```

**(45) Question 2.** We will use a Moore Finite State Machine, and 976.56 Hz real time interrupts (RTI).

(15) Part a) Show all global data structures required. In particular, define the simple circular linked list.

```
const struct State {
    unsigned char Out;           // Output to Port J
    const struct State *Next[2];}; // Next state depending on if input is 0, 1
typedef const struct State StateType;
#define S6A &fsm[0]
#define S6B &fsm[1]
#define S5A &fsm[2]
#define S5B &fsm[3]
#define S10A &fsm[4]
#define S10B &fsm[5]
#define S9A &fsm[6]
#define S9B &fsm[7]
StateType fsm[8]={
    {0x05, {S5A, S5B}, // State=S5A
    {0x05, {S6A, S5B}, // State=S5B
    {0x06, {S6A, S6B}, // State=S6A
    {0x06, {S10A, S6B}, // State=S6B
    {0x0A, {S10A, S10B}, // State=S10A
```

```

{0x0A, {S9A, S10B}, // State=S10B
{0x09, {S9A, S9B}, // State=S9A
{0x09, {S5A, S9B}}; // State=S9B
StatePtr *Pt; // Current State
(15) Part b) Show the ritual that initializes RTI, DDRJ, and data structures. Arm and enable interrupts.
void Ritual(void){
asm(" sei ");
    DDRJ=0x7F; // Unused pins are output
    RTICTL=0x81; // Arm, Set RTR to 1, 976.56Hz
    Pt=S5A; // initial state
    PORTJ=0x05;
asm(" cli ");}
(15) Part c) Show the real time interrupt ISR, which should be executed every 1024 μsec.
#pragma interrupt_handler RTIHan()
void RTIHan(void){
    RTIFLG=0x80; // Acknowledge by clearing RTIF
    if(PORTJ&0x80) // input from PJ7
        Pt=Pt->Next[1]; // next state if input=1
    else
        Pt=Pt->Next[0]; // next state if input=0
    PORTJ=Pt->Out; // output for this state
}

```

**(10) Question 3.** Is there a critical section? It depends on the assembly code generated by the compiler. If the compiler generates atomic operations, then no. This is the assembly code that both Hiware and ICC12 will produce.

```

SetBit1: bset PORTJ,#02
        rts
SetBit0: bset PORTJ,#01
        rts

```

If the compiler generates nonatomic read modify write operations, then yes

```

SetBit1: ldaa PORTJ
        oraa #02
        staa PORTJ
        rts
SetBit0: ldaa PORTJ
        oraa #01
        staa PORTJ
        rts

```