

(80) **Question 1.** Design the interface that receives data from a sensor.

(20) Part a) There are many answers but the basic idea is to tie **status** to two key wakeup lines. You can use any Port J key wakeup for the rising edge interrupt of **status**. The falling edge could use any Port H or Port D key wakeup interrupt. Any input port can be used for **data**. My answer uses

- **status** is tied to both PJ7 and PH7 to cause vectored interrupt on both rise and fall.
- **ack** is tied to PJ6 as a simple output.
- **data** is tied to Port T as eight simple inputs.

(30) Part b) Show the ritual that will be called at the beginning of the main program.

```
void Init(void) {
asm(" sei ");          // make ritual atomic
  DDRT=0x00;          // Port T are 8 inputs=data
  DDRH=0x00;          // PH7=status is an input
  KWIEH=0x80;         // arm keywakeup on fall on PH7=status (event 2)
  KWIFH=0x80;         // clear flag7
  DDRJ=0x40;          // PJ6=ACK is an output, PJ7=status is an input
  KPOLJ=0x80;         // keywakeup on rise on PJ7=status
  KWIEJ=0x80;         // arm keywakeup interrupt (event 4)
  KWIFJ=0x80;         // clear flag7
  InitFifo();
// fifo links background producer to foreground consumer
  PORTJ=0x40;         // PJ6=ACK is high, meaning get first data
asm(" cli ");         // enable interrupts
}
```

(15) Part c) Show the interrupt handler that is executed on the fall of status (timing event 2).

```
#pragma interrupt_handler KeyWakeupH()
void KeyWakeupH(void) { // event 2
  KWIFH=0x80;          // clear flag
  PutFifo(PORTT);      // read and save data
  PORTJ=0;             // ack=0 causing event 3
}
```

(15) Part d) Show the interrupt handler that is executed on the rise of status (timing event 4).

```
#pragma interrupt_handler KeyWakeupJ()
void KeyWakeupJ(void) { // event 4
  KWIFJ=0x80;          // clear flag
  PORTJ=0x40;          // ack=1 causing event 4
}
```

(20) **Question 2.** Consider the following simple C program

(10) Part a) Where in memory are the following objects allocated?

Object	global RAM 0x0800-...	stack RAM ...-0x0BFF	EEPROM 0xF000 to 0xFFFF
x1	xx		
x2	xx		
x3			xx
y1		xx	
y2		xx	
z1		xx	
z2		xx	
z3		xx	
add3			xx
main			xx

(10) Part b) Draw a stack picture at the point just after the addition in add3.

