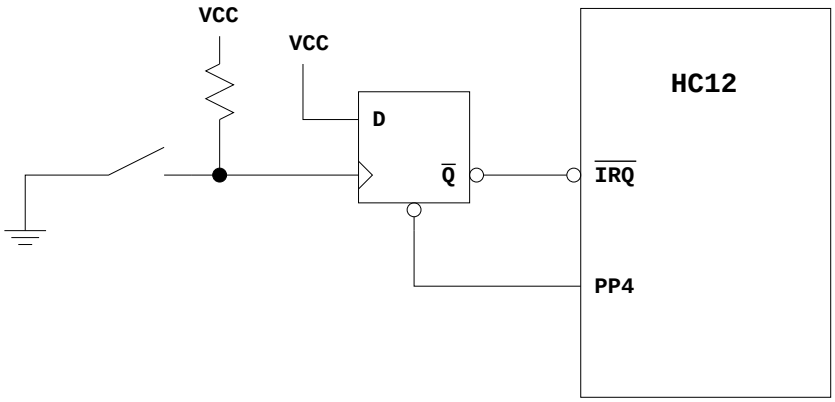
IRQ and XIRQ interrupts

- Lots of ways to interrupt the HC12
  - Timer Overflow
  - Real Time Interrupt
  - Input Capture
  - Output Comare
  - SPI
  - SCI
  - A/D
  - Others
- Two external pins just for interrupts: IRQ and XIRQ
- IRQ:
  - Enable with IRQEN bit of INTCR register
  - Mask with I bit of CCR
  - Can select falling-edge or low-level sensitivity with IRQE bit of INTCR register
- XIRQ:
  - Always enabled
  - Mask with X bit of CCR
  - Once masked, cannot unmask — non-maskable interrupt
- Special use of IRQ and XIRQ: get out of standby mode

WAIT and STOP

- HC12 has two low-power modes
- You enter the low-power mode with WAI and STOP instructions
- WAIT mode:
  - In WAIT mode current is reduced from max of 45 mA to max of 5 mA
  - You can program some subsystems to shut down or to continue operating in WAIT mode
  - Can tell Timer and A/D to shut down during wait mode
  - The more subsystems which shut down, the less power is consumed
  - The HC12 exits WAIT mode when an unmasked interrupt is received
- STOP mode:
  - In STOP mode current is reduced from max of 45 mA to max of 10 μA
  - You can enter STOP mode with the STOP instruction only if the S bit of the CCR is clear
  - All subsystems shut down in STOP mode
  - The HC12 exits STOP mode when an XIRQ or an unmasked IRQ is received (or RESET)
  - If XIRQ is masked when an XIRQ is received, the HC12 executes the next instruction after the STOP instruction
  - If XIRQ is unmasked when an XIRQ is received, the HC12 executes the XIRQ interrupt service routine



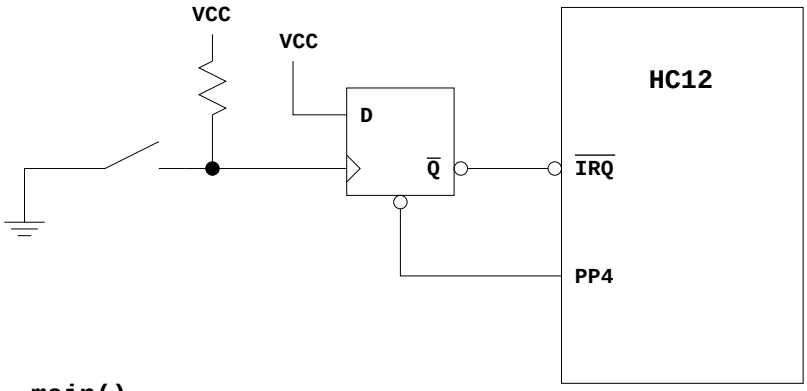

```
main()
{    DDRP = DDRP | 0x10;     /* Make PP4 output */
     PORTP = PORTP & ~0x10;  /* Reset flip-flop */
     PORTP = PORTP | 0x10;   /* Make FF active */
     INTCR = 0x40;           /* Level sensitive, enable IRQ */
     enable();               /* Clear I bit of CCR */
     while(1) _asm(" wai");  /* Stack registers, reduce power, wait for interru
}
@interrupt void irq_isr(void)
{
     Do what needs to be done on interrupt
     PORTP = PORTP & ~0x10;  /* Clear IRQ FF */
     PORTP = PORTP | 0x10;   /* Make FF active */
}
```

In WAIT mode, registers already stacked so HCA12 can get into ISR very quickly
Only way to get out of WAIT is an unmasked interrupt or RESET

In WAIT mode, CPU clock stops to save power.
Can save more power by turning off other resources not nedded
For example, can set TSWAI bit of TSCR to turn off timer subsystem in WAIT mode
can set SSWAI bit of SP0CR2 to turn off SPI clock in WAIT mode
can set ASWAI bit of ATDCTL2 to turn off A/D converter in WAIT mode
In WAIT mode with all subsystems off, power goes from 45mA to 5mA

S	X	H	I	N	Z	V	C	CCR

To use STOP mode you must clear S bit of CCR

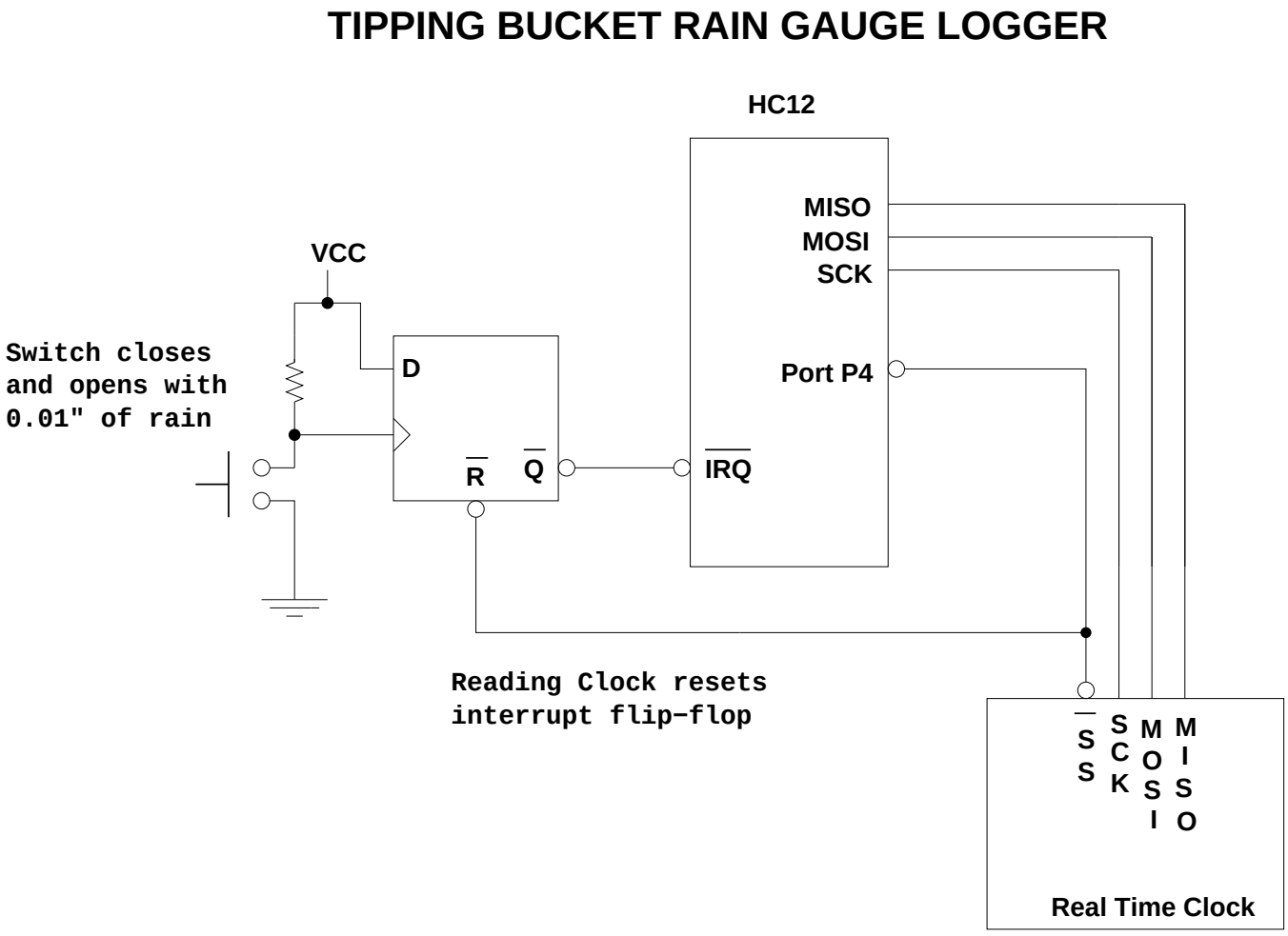



```
main()
{    DDRP = DDRP | 0x10;     /* Make PP4 output */
     PORTP = PORTP & ~0x10;  /* Reset flip-flop */
     PORTP = PORTP | 0x10;   /* Make FF active */
     INTCR = 0x40;           /* Level sensitive, enable IRQ */
     _asm(" andcc #$6f");    /* Clear S and I bits of CCR */
     while(1) _asm(" stop"); /* Stack registers, turn off clocks, wait for interru
}
@interrupt void irq_isr(void)
{
     Do what needs to be done on interrupt
     PORTP = PORTP & ~0x10;  /* Clear IRQ FF */
     PORTP = PORTP | 0x10;   /* Make FF active */
}
```

In STOP mode, registers already stacked so HCA12 can get into ISR very quickly
In STOP mode, all clocks stop to save power.
Only way to get out of STOP is RESET, XIRQ or IRQ
In STOP mode power reduced from 45mA to 10uA
At 10uA, HC12 can run for months on small battery

Rain Gauge

- Example of using the HC12 in STOP mode
- A switch is toggled whenever 0.01” of rain is detected
- System meant to run for three months on batteries
- Use an HC12 with a low-power real time clock
- At 10 μA, and 1000 mAh battery can run the HC12 for years
- The Real Time Clock is always running and consumes 0.5 mA of current
- With four D-cell batteries, system can run for three months



Go into stop mode while waiting for rain

On switch closure, HC12 wakes up, reads and records time, goes back to sleep

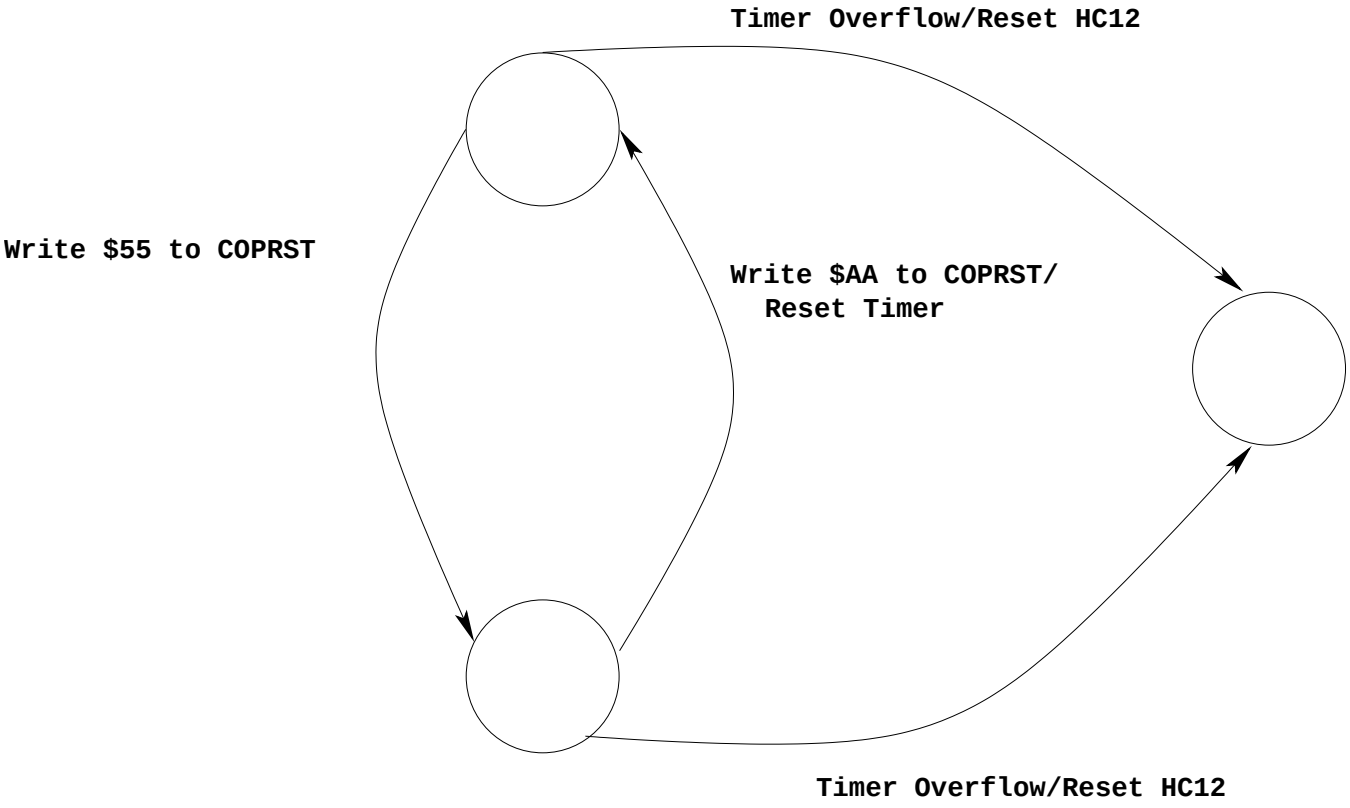
To read clock, HC12 brings PP4 (select for clock) low − this resets IRQ flip−flop

COP (Computer Operating Properly) Monitor

- When COP is enabled you have to write a pattern of bits to COPRST register within a set period of time
- You have to write a $55 followed by a $AA
- If you fail to do this, a COP failure reset exception occurs
- HC12 runs routine with interrupt vector $FFFA, $FFFB
- COP is set to time out after 1.024 ms by default
- You can change the timeout rate with bits 2, 1, and 0 of COPCTL register
- If you do not want to use COP monitor you must disable it by writing 0 the bits 2, 1, and 0 of COPCTL register

Clock Monitor

- When Clock Monitor is enabled a Clock Failure exception will be generated if the clock is running too slowly
- Any clock below 500 kHz is too slow
- You cannot use Clock Monitor is STOP mode because all clocks are stopped in STOP mode
- You enable clock monitor by setting the CME bit of COPCTL register



COP Timer Overflow Rate (CR2−0 of COPCTL)

000	Never
001	1.024 ms
010	4.096 ms
011	16.384 ms
100	65.536 ms
101	262.144 ms
110	524.288 ms
111	1,048.576 ms

Background Debug Mode (BDM)

- Background Debug Mode uses internal hardware to monitor what is happening inside the HC12 CPU
- For example can detect when the HC12 executes code outside of a specified address range
- The Background Debug Mode is used with another computer for debugging HC12 hardware and software
- Very powerful debugging tool
- Replaces very expensive external hardware debugging tools
- DBug-12 is set up so that your EVM can be used to control the BDM on another HC12

```
BACKGROUND DEBUG MODE

Hardware which monitors CPU and can stop CPU on some event

For example, can stop CPU when HC12 gets out of main program

Can figure out how HC12 got outside of program

To use background debug mode, must activate through its own
    serial communications interface.  This is normally done w
    another microprocessor
```