Help

The second interthread synchronization scheme is the **mailbox**. The mailbox is a binary semaphore with associated data variable. Interactive Tool 12.1 illustrates an input device interfaced using interrupt synchronization and uses a mailbox to send data from ISR to the main program. The mailbox structure is implemented with two shared global variables. **`Mail`** contains data, and **`Status`** is a semaphore flag specifying whether the mailbox is full or empty. The interrupt is requested when its trigger flag is set, signifying new data are ready from the input device. The ISR will read the data from the input device and store it in the shared global variable **`Mail`**, then update its **`Status`** to full. The main program will perform other calculations, while occasionally checking the status of the mailbox. When the mailbox has data, the main program will process it. This approach is adequate for situations where the input bandwidth is slow compared to the software processing speed. One way to visualize the interrupt synchronization is to draw a state versus time plot of the activities of the hardware, the mailbox, and the two software threads .

Use the following tool to how the foreground and background thread communicate using a "mailbox"

start pause reset

Enter 4-digit hex Data (in the format 0x**):**

Input Data

Using the tool demonstrates that during execution of block A, the mailbox is empty, the input device is idle and the main program is performing other tasks, because mailbox is empty. When new input data are ready, the trigger flag will be set, and an interrupt will be requested. In block B the ISR reads data from input device and saves it in **`Mail`**, and then it sets **`Status`** to full. The main program recognizes **`Status`** is full in Block C. In Block D, the main program processes data from **`Mail`**, sets **`Status`** to empty. Notice that even though there are two threads, only one is active at a time. The interrupt hardware switches the processor from the main program to the ISR, and the return from interrupt switches the processor back.

The third synchronization technique is the **FIFO queue**. The use of a FIFO is similar to the mailbox, but allows buffering, which is storing data in a first come first served manner. For an input device, an interrupt occurs when new input data are available, the ISR reads the data from the input device, and puts the data in the FIFO. Whenever the main program is idle, it will attempt to get data from the FIFO. If data were to exist, that data will be processed. The big arrows in Figures 12.4 and 12.5 signify the communication and synchronization link between the background and foreground.

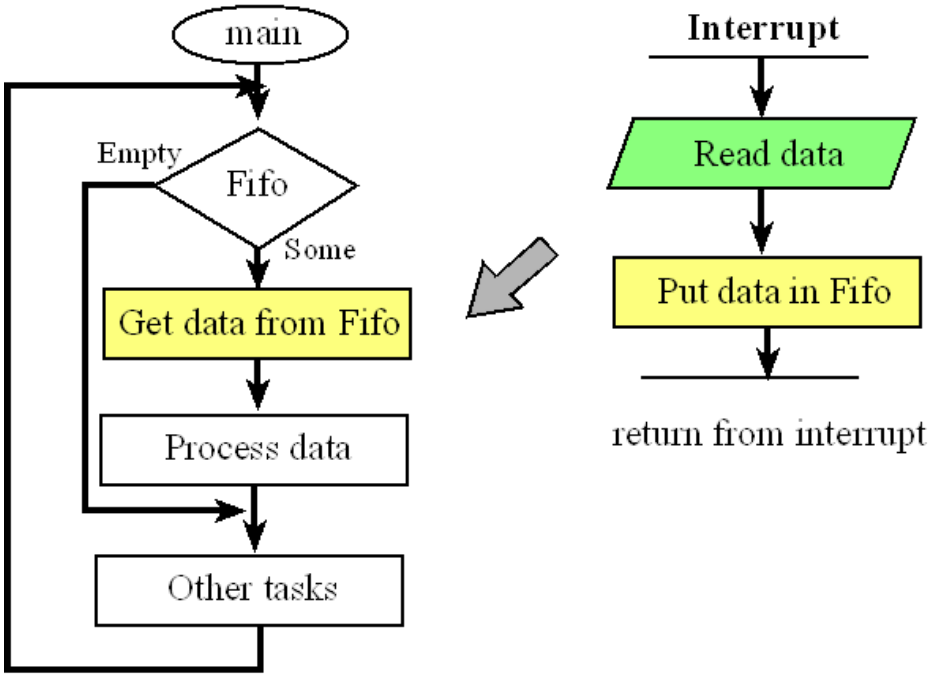

Figure 12.2. For an input device we can use a FIFO to pass data from the ISR to the main program.

For an output device, the main program puts data into the FIFO whenever it wishes to perform output. This data is buffered, and if the system is properly configured, the FIFO never becomes full and the main program never actually waits for the output to occur. An interrupt occurs when the output device is idle, the ISR gets from the FIFO and write the data to the output device. Whenever the ISR sees the FIFO is empty, it could cause the output device to become idle. The direction of the big arrows in Figures 12.2 and 12.3 signify the direction of data flow in these buffered I/O examples.

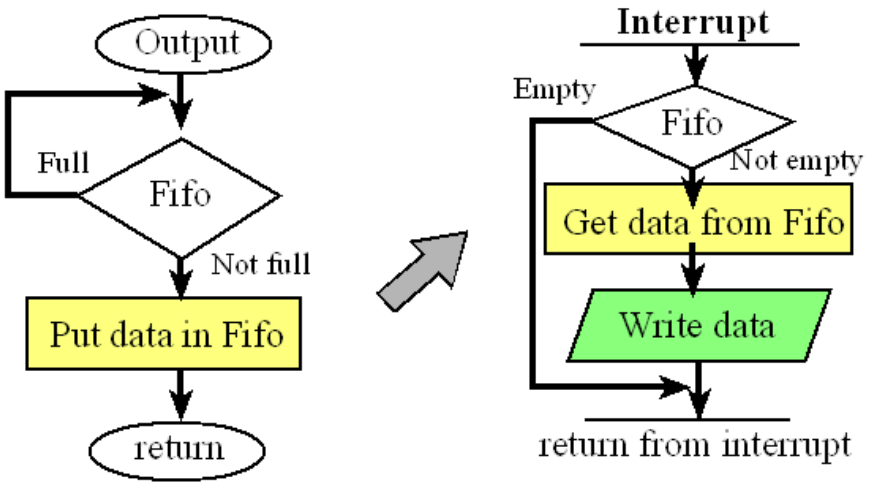

Figure 12.3. For an output device we can use a FIFO to pass data from the main program to the ISR.

Performance Tip: It is poor design to employ backward jumps in an ISR, because they may affect the latency of other interrupt requests. Whenever you are thinking about using a backward jump, consider redesigning the system with more or different triggers to reduce the number of backward jumps.

Help