AVRfreaks DESIGN NOTE #029

AUTHOR: CHRISTOPH REDECKER - BUFFI.GANGZTA-LAND@T-ONLINE.DE

KEYWORDS: SRAM, BUFFERS, FIFO

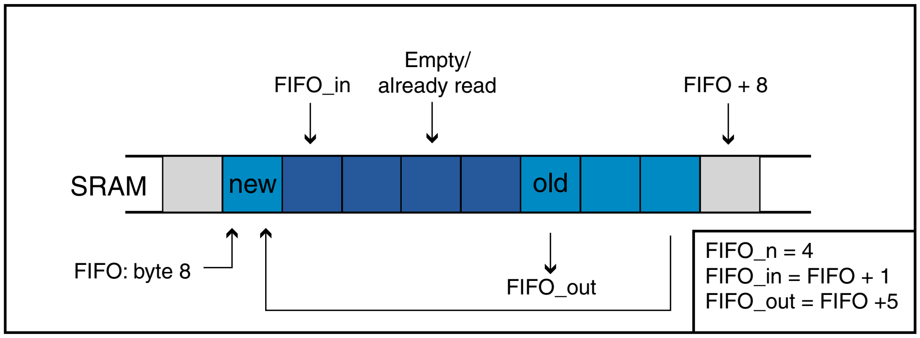
Implementing FIFO Buffers in Software

It's not always possible (or wanted) to handle data byte for byte. If the application code contains a software FIFO (First In – First Out) buffer, data can be handled in packets, or whole strings can be processed in one go. There are many ways of how to implement a FIFO buffer in software, and this is how I do it:

The buffer is basically a RAM segment holding the data. Pointers are used to write new data and read old data.

When data is received, it is stored at the end of the buffer and the number of bytes in the buffer is increased, reading from the buffer is done at the beginning of the buffer and the number of bytes in the buffer is decreased. This is done by having two pointers relating to these locations (FIFO_in: end, FIFO_out: beginning) and a counter variable (FIFO_n) holding the amount of data stored. If a byte is to be added, it is written to SRAM and the pointer is increased. Same for reading (amount is decreased). Below is a simple drawing showing the SRAM segment used for buffering data, the pointers and what is done when the end of the buffer RAM is reached (the pointers roll over to the buffer's beginning).

Figure 1. 8-bit FIFO Buffer



In this example the buffer is set up for eight bytes width. If a byte is to be read, this is basically done by

```
ld data, FIFO_out+
```

FIFO_out is an index register pair such as X, Y, and Z. If this is repeated three times (see example drawing), FIFO_out will point at FIFO + 8 (FIFO + FIFO_length) and roll over to the beginning of the RAM segment. One byte can still be read (FIFO_n - 3 = 1). FIFO_in stores data at locations that have already been read by the application. The application should periodically check if the buffer is full or contains enough data. This is done by checking FIFO_n against FIFO_length or any lower number between 0 and FIFO_length.

The Code

Note: This is an example for the UART (RX). If it is used within an ISR (UART RX complete) dedicated registers may have to be used depending on the individual application.

```
**************************************************
fifo SRAM space:
save_y:        .byte 2             ; to save Y register pair

rxfifo_length:                     ; length or size of fifo buffer

rxfifo_base:   .byte rxfifo_length ; reserves fifo_length bytes for the
                                   ; buffer and is used as the buffer's
                                   ; base address

rxfifo_n:      .byte 1             ; number of bytes currently stored in
                                   ; buffer (init: 0)
rxfifo_in:     .byte 2             ; pointer to address in buffer written
                                   ; to (init: rxfifo_base)
rxfifo_out:    .byte 2             ; pointer to address in buffer read
                                   ; from (init: rxfifo_base)
IO_source:                         ; source where the byte to be stored
                                   ; is read from. Can also be a register
                                   ; source if needed (for example
                                   ; UART_TXC)
data_out:                          ; register where data from buffer is
                                   ; stored, can also be an I/O
                                   ; destination
```

temp2 has to be a high order register (16..31).

```
;**************************************************
add_rxFIFO:
  in     temp, IO_source           ; get data to be stored
  lds    temp2, rxfifo_n           ; check if buffer full
  cpi    temp2, rxfifo_length
  breq   end_add_rxfifo            ; if so, end (change
  sts    save_Y, YL                ; this to overwrite old data)
  sts    save_Y + 1, YH            ; save Y reg pair
  lds    YL, rxfifo_in             ; setup FIFO_in
  lds    YH, rxfifo_in + 1
  st     Y+, temp                  ; store data
  lds    temp, rxfifo_n            ; inc FIFO_n
  inc    temp
  sts    rxfifo_n, temp
  cpi    YL, low(rxfifo_base + rxfifo_length)
                                   ; 16-bit cpi with FIFO_base +
  ldi     temp2, high(rxfifo_base + rxfifo_length)
                                    ; FIFO_length
  cpc     YH, temp2
  breq    add_rxfifo_rollover       ;if end of buffer ram
  end_add_rxfifo:                   ;reached, roll over
  sts     rxfifo_in, YL             ;store FIFO_in
  sts     rxfifo_in + 1, YH
  lds     YL, save_Y                ;restore Y reg pair
  lds     YH, save_Y + 1
  ret                               ;return
  add_rxfifo_rollover:
  ldi     YL, low(rxfifo_base)      ;load FIFO_in with
  ldi     YH, high(rxfifo_base)     ;FIFO_base
  rjmp    end_add_rxfifo
;**************************************************
get_rxfifo:                         ;works just like add_FIFO
  lds     temp, rxfifo_n
  tst     temp
  breq    end_get_rxfifo
  sts     save_Y, YL
  sts     save_Y + 1, YH
  lds     YL, rxfifo_out
  lds     YH, rxfifo_out + 1
  ld      data_out, Y+
  lds     temp, rxfifo_n
  dec     temp
  sts     rxfifo_n, temp
  cpi     YL, low(rxfifo_base + rxfifo_length)
  ldi     temp2, high(rxfifo_base + rxfifo_length)
  cpc     YH, temp2
  breq    get_rxfifo_rollover
  end_get_rxfifo:
  sts     rxfifo_out, YL
  sts     rxfifo_out + 1, YH
  lds     YL, save_Y
  lds     YH, save_Y + 1
  ret
  get_rxfifo_rollover:
  ldi     YL, low(rxfifo_base)
  ldi     YH, high(rxfifo_base)
  rjmp    end_get_rxfifo
;**************************************************
```

If you want a TX fifo buffer, the ISR has to read data from the buffer and thus has to rcall get_fifo (or get_txfifo or whatever you call it). Remember to change the internal labels accordingly or the assembler will give you error warnings. If you need both an RX and a TX buffer, you need routines for both and also double ram space (save_y can be shared). Here are example ISRs for both RX complete and TX complete:

```
UART_RXC:
  in      itemp, SREG
  push    itemp
  rcall   add_rxfifo
  pop     itemp
  out     SREG, itemp
  reti

UART_DRE:
UART_TXC:
  in      itemp, SREG
  push    itemp
  cbi     UCR, UDRIE
  rcall   get_txfifo
  pop     itemp
  out     SREG, itemp
  reti
```

If you use this example ISR, get_txfifo also has to move the data to UDR.

```
ld      data_out, Y+
out     UDR, data_out               (<--add this line)
```

Usage of UART_DRE

If you just filled the buffer, you can start a transmisson in two ways: Either call UART_TXC or get_txfifo or just set UDRIE in UCR. A transmission will start immediately and UDRIE in UCR will be cleared by the ISR to prevent write collisions. Then, every time a character has been sent, UART_TXC will be called provided that TXCIE in UCR is set.

When no data is available (everything has been sent), no new data is written to UDR and no new interrupt occurs until UDRIE is set again.

Only set UDRIE if the buffer was empty before! If your code detects that txfifo_n is 0, you can fill the buffer and then set UDRIE, but if there's still data in the buffer, setting UDRIE can result in write collisions. Just keep on filling the buffer instead, your data will be sent after old data is out. If the buffer is full, attempting to write to the buffer will NOT add it, as add_fifo checks if the buffer is full.

I hope this helped at least some of you, especially those who never coded a FIFO buffer before. If you want, you can add more features like full/empty flags, error logging etc. You can even combine RX/TX buffers to have an SPI buffer! It's up to you.