STANDARD
MICROSYSTEMS
CORPORATION

APPLICATION NOTE 7.2

GPIO Expansion for the FDC37N95xFR

By Nizar Azzam

This document is targeted for board designers who require more GPIOs than the provided GPIOs in a given FDC37N95xFR part. The goal is to identify different designs to expand the GPIOs. Design simplicity/complexity varies with the required expanded GPIO's flexibility. For example, the design would be very simple if only eight expanded GPIOs are required and used as outputs only. On the other hand, design gets more complex if more than eight input and output GPIOs are required. A general purpose pin (GPIOx) is used as a chip select and for the sake of clarity this document renames GPIOx to GPCS.

The tables and diagrams below demonstrate design options to expand the GPIOs. It is very important to understand that in order to add the flexibility of reading the expanded GPIOs, designer must add external decoding logic and run code from internal RAM.

A pull up resistor on GPCS is used to enable reading FLASH at power up. This is very important since at power up, the operating system needs to shadow BIOS from FLASH to main memory before running any code to configure the GPIOs. When GPCS equals '0', this pin is used to access the expanded GPIOs (8051 must be running from internal code when reading GPIOs). When GPCS equals '1', FLASH is enabled and code can be fetched.

The MOVX instruction can be used to allow reading and writing the expanded GPIO port. The expanded GPIO port is accessed whenever any read or write cycle using the MOVX instruction to address 8000h-FFFFh (8000h-FFFFh is for port expansion), and GPCS is active before, during, and after the RD/WR strobe. If the GPCS=1, then it is a cycle to the Flash.

Sequence of Operation

Write to expanded GPIOs:

1. Fetch code from FLASH.
2. Store fetched data in internal scratch RAM (0x7D00h-0x7E00h).
3. Repeat steps 1 and 2 until all necessary code to run "GPIO Write" operation is stored in RAM.
4. Store GPIO_OUT value in the accumulator.
5. Set MMC (0x7FF4h bit 3) to 1. This makes Scratch RAM (0x7D00h-0x7E00h) look like Scratch ROM (0x00h-0xFFh).
6. Store program pointers.
7. Jump to "GPIO write" code which is stored in scratch ROM.
   a) Set GPCS pin to logic low to access expanded GPIOs.
   b) Start executing code to write to the expanded GPIOs.
8. Read stored program pointers.
9. Resume executing code from flash.

80 Arkay Drive
Hauppauge, NY 11788
(516) 435-6000
FAX (516) 273-3123

Read From expanded GPIOs:

1. Fetch code from FLASH.
2. Store fetched data in internal scratch RAM.
3. Repeat steps 1 and 2 until all necessary code to run “GPIO Read” operation is stored in RAM.
4. Set MMC (0x7FF4h bit 3) to 1. This makes Scratch RAM (0x7D00h-0x7E00h) look like Scratch ROM (0x00h-0xFFh).
5. Store program pointers.
6. Jump to “GPIO Read” code from internal RAM.
   a) Set GPCS pin to logic low to access expanded GPIOs.
   b) Start executing code to read from the expanded GPIOs.
7. Read stored program pointers.
8. Resume executing code from flash.

a)

Other methods to expand GPIOs :

1- Use a PAL to implement the external address decoding.
2- Use one of the most significant address bits and a GPIO to memory map the GPIOs.

Tables and Figures Description:

Table 1 and Figure 1 : Expanded GPIO Write Only

These table and Figure illustrate the use of ONE GPIO. This approach is very simple to implement in the sense of hardware and software. The limitation is that expanded GPIOs can be used as outputs only. The 8051 can fetch code from FLASH and write to GPIOs without running from internal RAM.

Table 2 and Figure 2 : Expanded GPIO Read/Write Using External Gates

These table and Figure illustrate the use of one GPIO and some external logic gates. This approach is more involved than the previous in the sense that the 8051 must shadow code to the internal RAM, and run code from internal RAM to access the expanded GPIOs. (See Sequence of Operation for more details)

Table 3 and Figure 3 : Expanded GPIO Read/Write Using External Gates And Address Decoders

These table and Figure illustrate the use of ONE GPIO, external logic gates, and address decoders. This approach is more involved then the previous two in the sense that the 8051 must shadow code to the internal RAM (Use ADDRESS[2:0] to select an expanded GPIO bank),and run code from internal RAM to access the expanded GPIOs. (See Sequence of Operation for more details). The advantage of this method is the flexibility of reading and writing the expanded GPIOs and providing large number of GPIOs.

Table 1 : Expanded GPIO Write Only

nRD	nWR	GPCS	nFCS	nGCS	ACCESS TYPE
1	1	x	1	1	Don't care
0	0	x	0	GPCS	Not Valid
1	0	0	1	nCSO = 0	Write Expanded GPIOs
0	1	0	1	1	Read Flash
1	0	1	0	1	Write Flash
0	1	1	0	1	Read Flash

Table 2 : Expanded GPIO Read/Write Using External Gates

nRD	nWR	GPCS	nFCS	nGCS	ACCESS TYPE
1	1	x	1	1	Don't care
0	0	x	nGPIO	GPCS	Not Valid
1	0	0	1	nCSO =0	Write Expanded GPIOs
0	1	0	1	nCSI = 0	Read Expanded GPIOs
1	0	1	0	1	Write Flash
0	1	1	0	1	Read Flash

Table 3 : Expanded GPIO Read/Write Using External Gates And Address Decoders

nRD	nWR	GPCS	nFCS	nGCS	ACCESS TYPE
1	1	x	1	1	Don't care
0	0	x	nGPIO	GPCS	Not Valid
1	0	0	1	nCSO[CBA] =0	Write Expanded GPIOs
0	1	0	1	nCSI[CBA] = 0	Read Expanded GPIOs
1	0	1	0	1	Write Flash
0	1	1	0	1	Read Flash

Figure 1 : Expanded GPIO Write Only

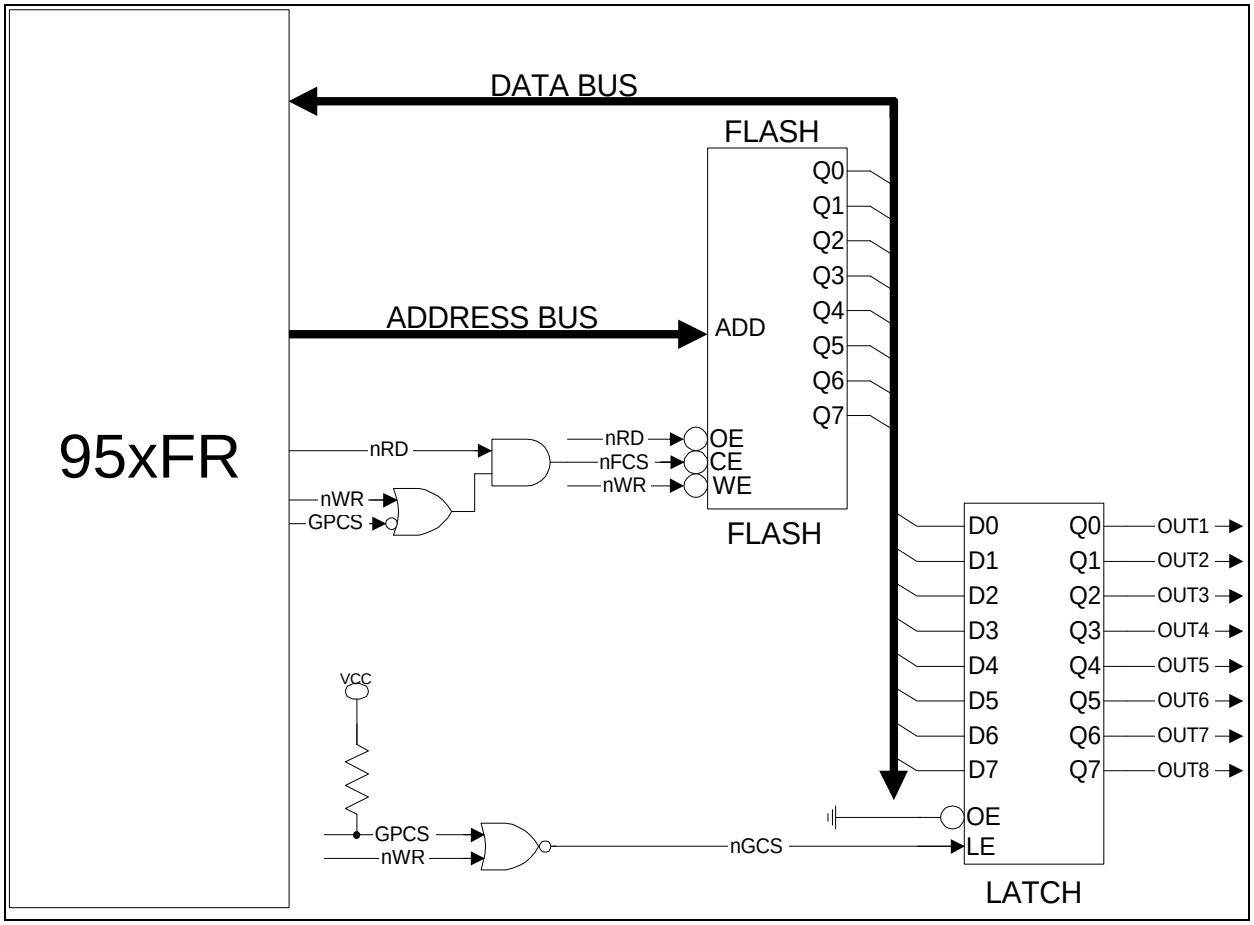

Figure 2 : Expanded GPIO Read/Write Using External Gates

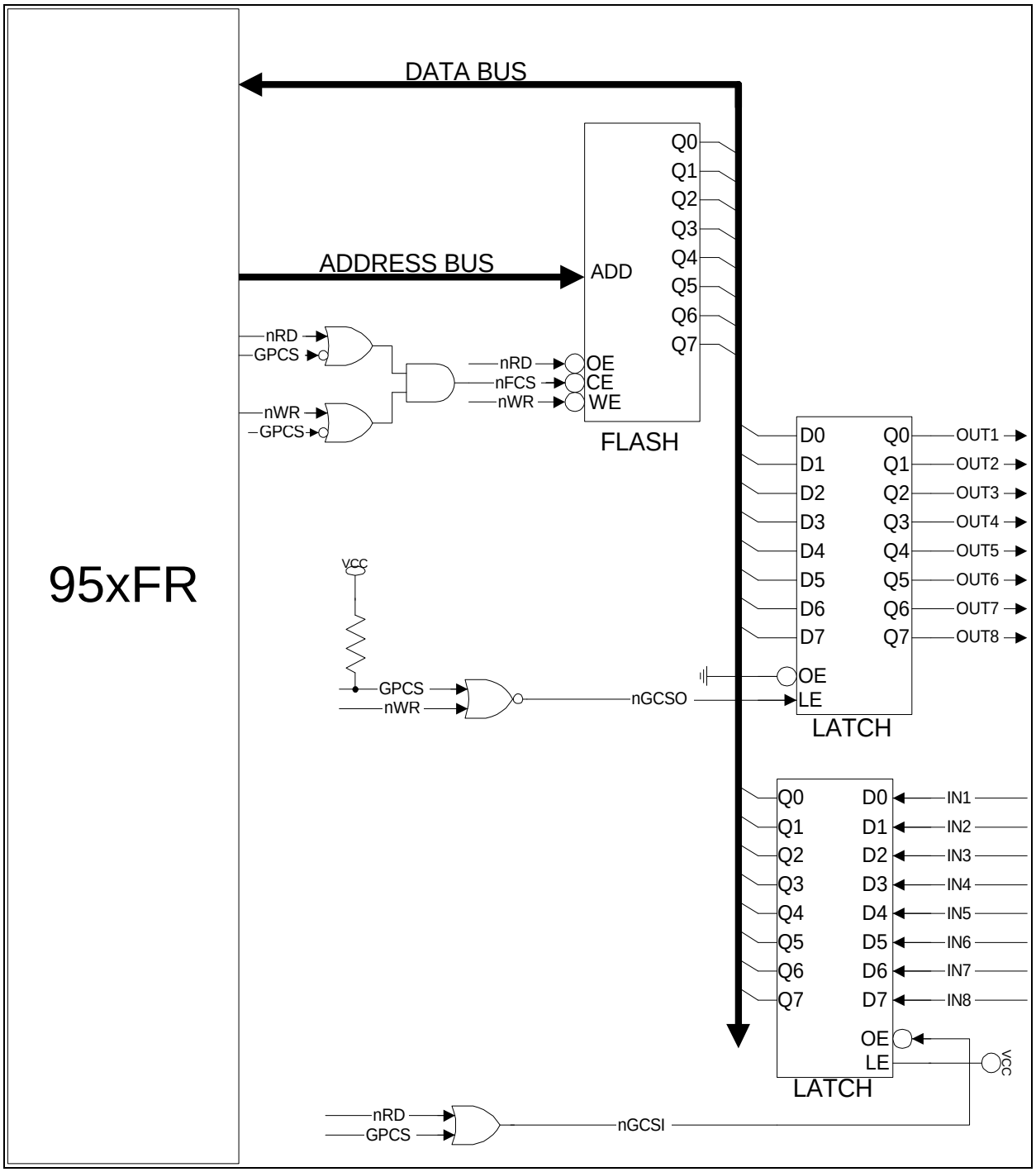

Figure 3 : Expanded GPIO Read/Write Using External Gates And Address Decoders