Single stepping through the GPIO write instruction

`GPIO_PORTF_DATA_R |= R_LED;` we see that the data_F register is initialized and it's address stored in R0 register. The LDR R1, [R0] is used to access the data stored at address pointed to by R0 (our data_F register) and is saved in R1. This data is then OR'd with the bits needed to turn on the red_led (0010). The final instruction STR R1, [R0] stores the OR'd result in R1 to the memory location pointed to by R0.

```
        0x66: 0x9000          STR      R0, [SP]
  GPIO_PORTF_DATA_R |= R_LED;          //setting GPIO_DATA..
        0x68: 0x480f          LDR.N    R0, [PC, #0x3c]    ..
        0x6a: 0x6801          LDR      R1, [R0]
        0x6c: 0xf051 0x0102   ORRS.W   R1, R1, #2
        0x70: 0x6001          STR      R1, [R0]
        0x72: 0xe002          B.N      ??main_2           ..
    count++;
```

Symbolic Memory

Go to: 0x400253fc

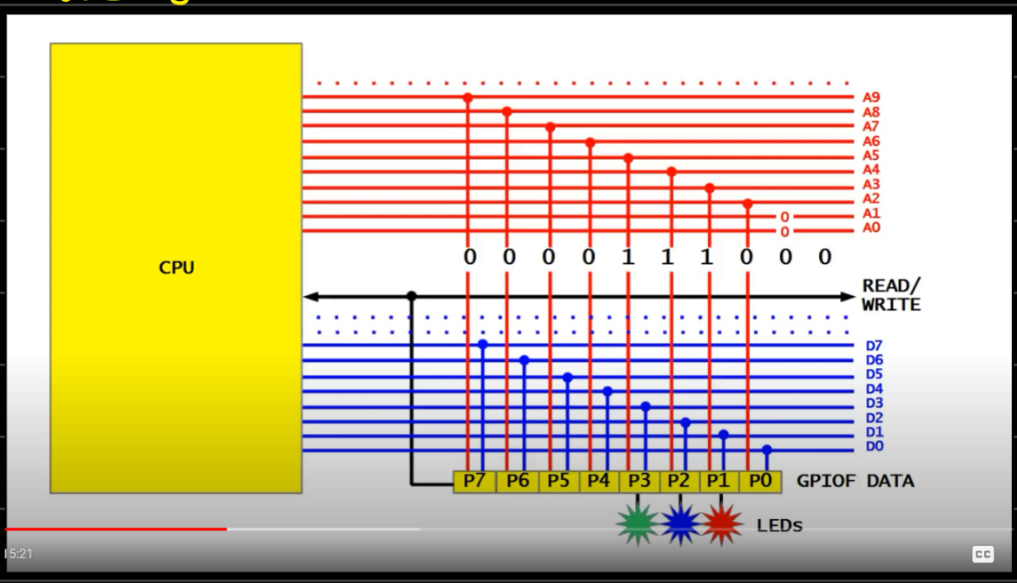
Location	Data
0x4002'53f0	0x0
0x4002'53f4	0x0
0x4002'53f8	0x2
0x4002'53fc	0x2
0x4002'5400	0xe
0x4002'5404	0x0

→ data_F register is now 0x02 (red)

This is a read modify write and all pins are accessed by a single address.

However, we can access each bit in a data register using unique addresses (this means we don't disturb any other GPIO bits). This is also important because any interrupt in-between a read-modify-write sequence can result in our read-modify-write sequence not working correctly. Thus we need to find alternatives to read-modify-writes.

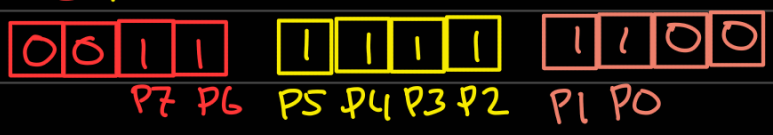
Wiring between GPIO data register, CPU and address bus:



P1, P2, P3 can get data from data register only if their connected address lines are also selected (1). A1 and A0 not used (address must be divisible by 4).

The 8 GPIO bits can be of any combination and thus are given 256 (2^8) 32 bit registers to access each GPIO-pin in a bitwise fashion. The starting address for this range of registers is 0x40025000. In our previous code (week 6) we used 0x400253FC for data_F register access. This allowed all GPIO data lines (P0-P7) to be active:

3FC



In ARM this bit wise manipulation is provided by setting the address bits 9:2 of the data register corresponding to the pins we want to use. Recall:

10.2.1.2 Data Register Operation

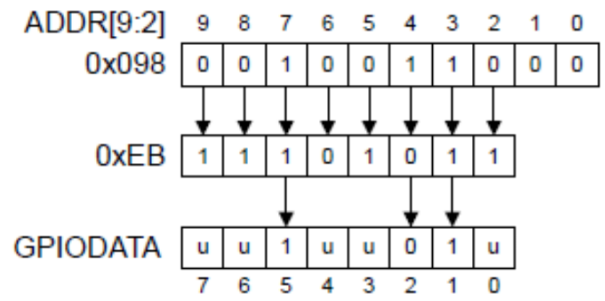
To aid in the efficiency of software, the GPIO ports allow for the modification of individual bits in the **GPIO Data (GPIODATA)** register (see page 662) by using bits [9:2] of the address bus as a mask. In this manner, software drivers can modify individual GPIO pins in a single instruction without affecting the state of the other pins. This method is more efficient than the conventional method of performing a read-modify-write operation to set or clear an individual GPIO pin. To implement this feature, the **GPIODATA** register covers 256 locations in the memory map.

During a write, if the address bit associated with that data bit is set, the value of the **GPIODATA** register is altered. If the address bit is cleared, the data bit is left unchanged.

For example, writing a value of 0xEB to the address GPIODATA + 0x098 has the results shown in Figure 10-3, where u indicates that data is unchanged by the write. This example demonstrates how **GPIODATA** bits 5, 2, and 1 are written.

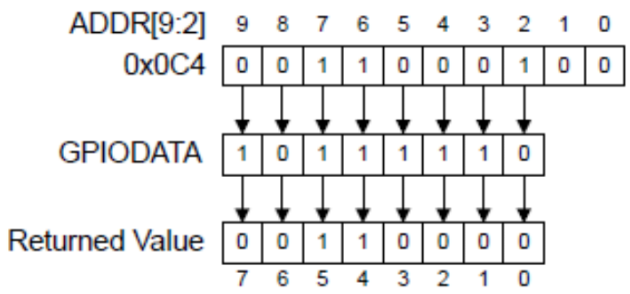
The 256 locations to access each bit combination of the 8 GPIO pins.

Figure 10-3. GPIODATA Write Example



During a read, if the address bit associated with the data bit is set, the value is read. If the address bit associated with the data bit is cleared, the data bit is read as a zero, regardless of its actual value. For example, reading address GPIODATA + 0x0C4 yields as shown in Figure 10-4. This example shows how to read **GPIODATA** bits 5, 4, and 0.

Figure 10-4. GPIODATA Read Example



This bit wise access is enabled by using the pointer GPIO_DATA_BITS_R in the header file and using pointer arithmatic (in hexadecimals) to access individual pins connected to GPIO data register.

APB vs AHB AHB is faster than APB. To select AHB of a port we need to set the bit field of that port in the GPIOHBCTL register

GPIO High-Performance Bus Control (GPIOHBCTL)

Base 0x400F.E000
Offset 0x06C
Type RW, reset 0x0000.7E00

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										PORTF	PORTE	PORTD	PORTC	PORTB	PORTA
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

↑ here to enable port F access through AHB

Bit/Field	Name	Type	Reset	Description
31:6	reserved	RO	0x0000.0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
5	PORTF	RW	0	Port F Advanced High-Performance Bus This bit defines the memory aperture for Port F. Value Description 0 Advanced Peripheral Bus (APB). This bus is the legacy bus. 1 Advanced High-Performance Bus (AHB)
4	PORTE	RW	0	Port E Advanced High-Performance Bus This bit defines the memory aperture for Port E. Value Description 0 Advanced Peripheral Bus (APB). This bus is the legacy bus. 1 Advanced High-Performance Bus (AHB)

*register names will also change when using AHB.