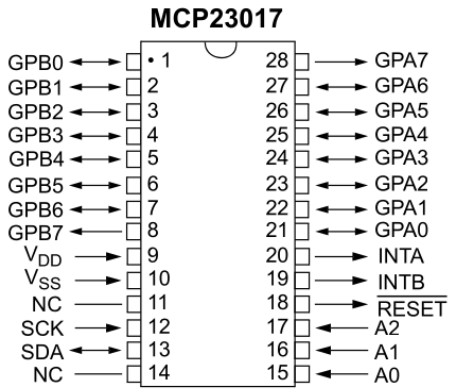
The MCP23017 as input port expander

The MCP23017 is used by Zynthian so that more buttons can be connected.
As I had some problems with this I wanted to test the port expander independantly of Zynthian.
To do this I connected the MCP23017 to the I2C bus of a Raspi Pico on which the test program was executed.

Pinout and register information



- The address pins select the I2C address. It is 0x20 when A0, A1, A2 are connected to GND.
- Pin 10 (Vss) is connected to GND
- Pins 18 (Reset\) and Pin 9 (VDD) are connected to +3.3V

There are 22 registers (11 for each port)

At power up the state is:

[255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Looking at the data sheet, this means that ports A (and B as we will see) are inputs.

There is a complication: there are 2 possible configurations depending on the IOCON.BANK bit.

This is really confusing! You must read IOCON to get the bank mode. But depending on the bank mode IOCON has a different address! This could have been realised otherwise, causing less pain…

Fortunately at power up we see that all registers except registers 0 and 1 are 0. This means that IOCON.BANK = 0 and we have bank mode 0.

For this mode the table in the datasheet gives these addresses:

TABLE 3-5: CONTROL REGISTER SUMMARY (IOCON.BANK = 0)

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALA	06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
DEFVALB	07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONA	08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
INTCONB	09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
IOCON	0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
INTFA	0E	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTFB	0F	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTCAPA	10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
INTCAPB	11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

If we connect buttons or encoders it makes sense to use the internal pullup resistors.

This is done by writing 0xFF to the registers GPPUA and GPPUB at addresses 0x0C and 0x0D.

Inputs can be read at addresses 0x12 (port A) and 0x13(port B).

Interrupts are enabled by setting the bits in the GPINTENA / GPINTENB register. When the IOCON register has all 0, as it has after startup, the interrupt polarity is active low, and every port generates its own interrupt.

Normally when an interrupt occurs the interrupt flag should be cleared, but for testing purposes (measuring the INTA/B output with the oscilloscope) it is OK to do nothing. The output goes high automatically and we see a negative pulse on every input change.

Testing with a Raspi Pico

The port expander is connected to the Pico's SCL and SDA with 3.3K pullup resistors.

```
# Test MCP23017 as input port expander
# connected to I2C1

from machine import Pin, I2C
```

```
import time

# Scan addresses
print("External devices on I2C port 1, GPIO27(SCL), GPIO26(SDA)")
i2c = I2C(1, scl=Pin(27), sda=Pin(26), freq=100000)
addresses = i2c.scan()
for a in addresses:
    print(hex(a))

# Rudimentary class for the port expander
class MCP23017():

    def __init__(self, i2c, mcp_addr=0x20):
        self.addr = mcp_addr
        self.i2c = i2c

    def read_regs(self):
        # read all 22 registers and return list of values
        x = self.i2c.readfrom_mem(self.addr, 0, 22)
        x = list(x)
        return x

    def write_reg(self, reg_addr, value):
        # write value into reg_addr
        self.i2c.writeto_mem(self.addr, reg_addr,  bytes([value]))

    def pullup_all(self):
        # Set pullups 100k for A and B port
        self.write_reg(0xc, 0xFF)
        self.write_reg(0xd, 0xFF)

    def read_A(self):
        # Read 8 bits on port A
        x = self.i2c.readfrom_mem(self.addr, 0x12 , 1)
        return x[0]

    def read_B(self):
        # Read 8 bits on port B
        x = self.i2c.readfrom_mem(self.addr, 0x13 , 1)
        return x[0]

    def enable_int_A(self):
        self.write_reg(0x04, 0xFF)

    def enable_int_B(self):
        self.write_reg(0x05, 0xFF)

#-------------------------------------------------------
# TEST
mcp = MCP23017(i2c)

print("Register values after powerup")
x = mcp.read_regs()
print(x)

print("Register values after switching on pullups:")
mcp.pullup_all()
x = mcp.read_regs()
print(x)

print("Enabling interrupts")
mcp.enable_int_A()
mcp.enable_int_B()

print("Reading A and B ports in loop, stop with Ctrl-C")
input("Start = <Enter>")

while True:
    a = mcp.read_A()
    b = mcp.read_B()
    print(hex(a),hex(b))
    time.sleep(0.05)
```