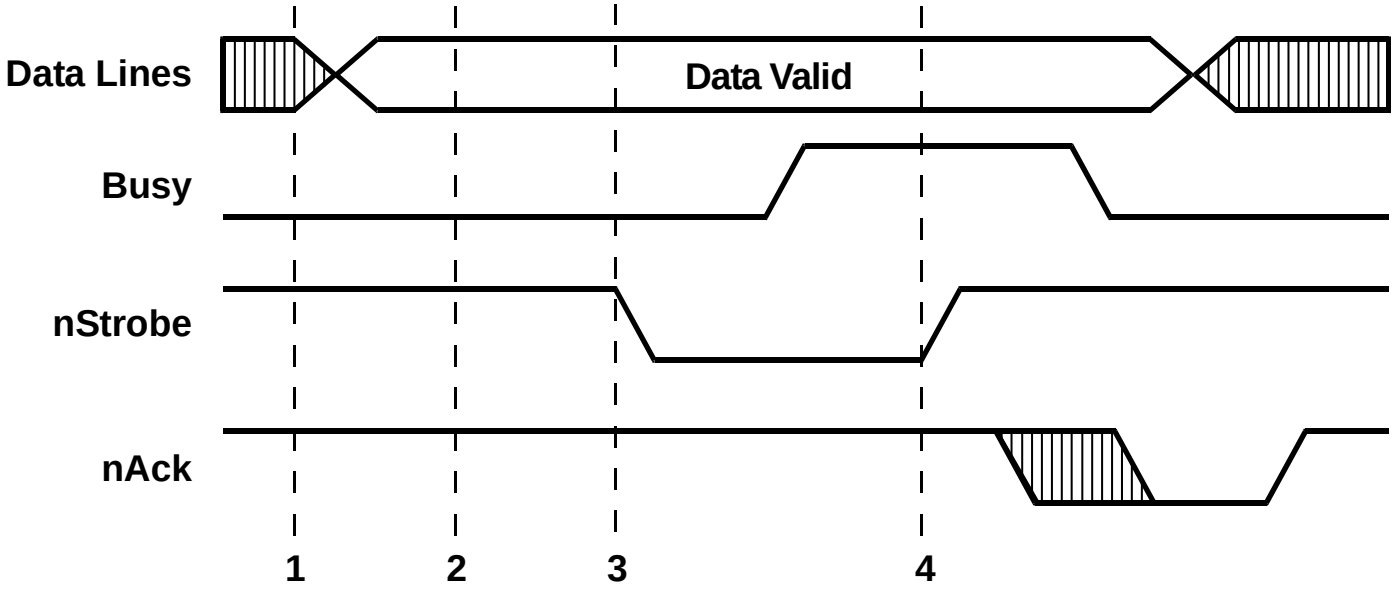
Parallel Interface

- Parallel ports and LCD
- DC Motor and Relay
- Stepper motor
- Camera
- ISA Bus
- PCI Bus
- ARM Bus
- Other buses

Parallel port

- Multi-bit I/O
- Short distances

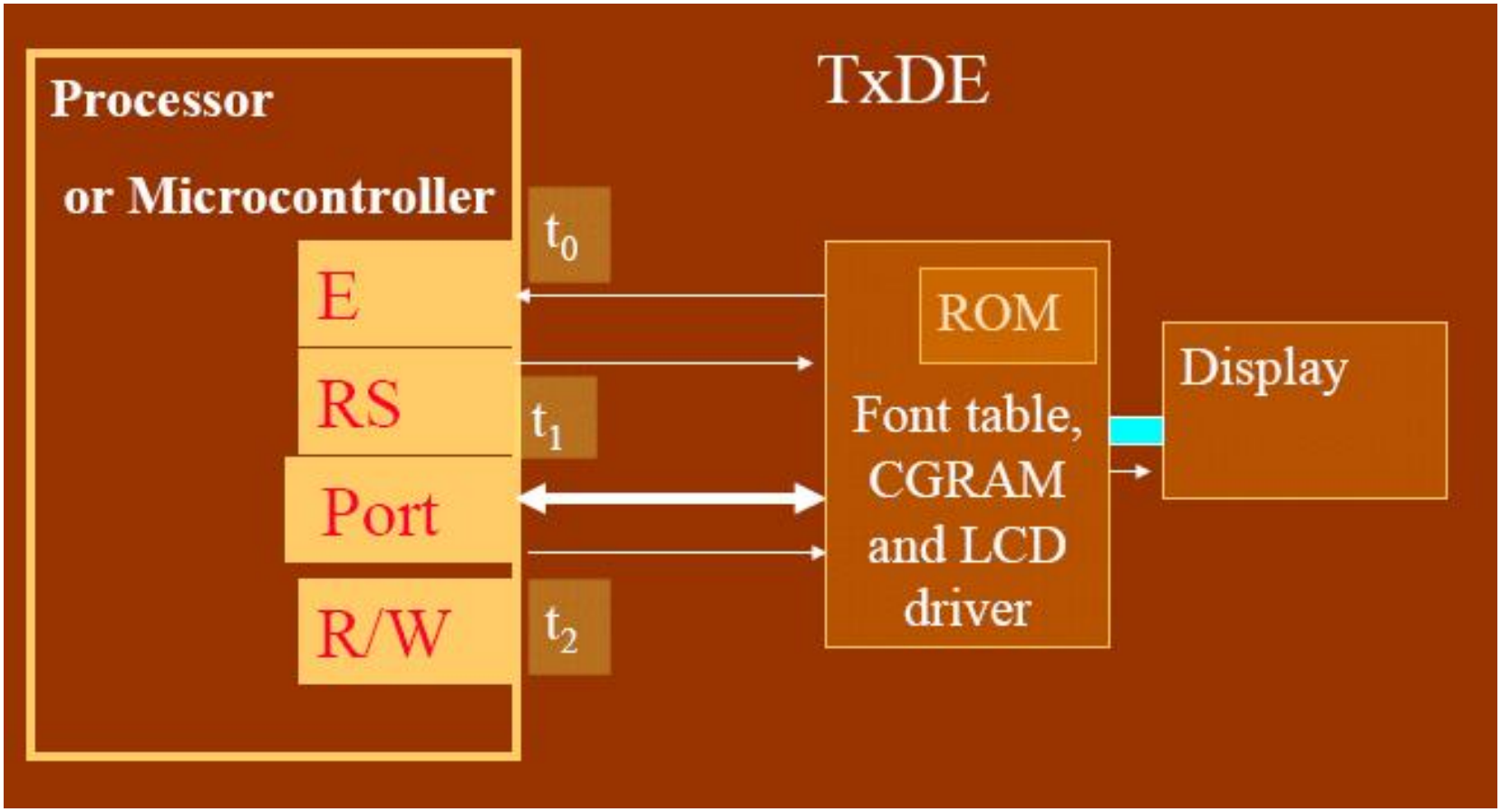
Parallel Port transfer of an 8-bit data value to a printer



Parallel Printer I/O Operation

- 1. Computer waits for port's output ready bit (i.e. busy=0)
- 2. Computer outputs a new 8-bit data value
- 3. Computer toggles strobe control line
- 4. Computer must wait for ack to pulse low before any data or strobe line changes
- Printer can be busy for long time periods
  (out of paper, offline, page feed…etc)

LCD controller interface



LCD

- RS is reset signal
- E is enable bit

  There is an interval in which controller maybe in disable state such as clear display (e.g., takes 150 μs)
- Port- 8 bit parallel port for control/data bus

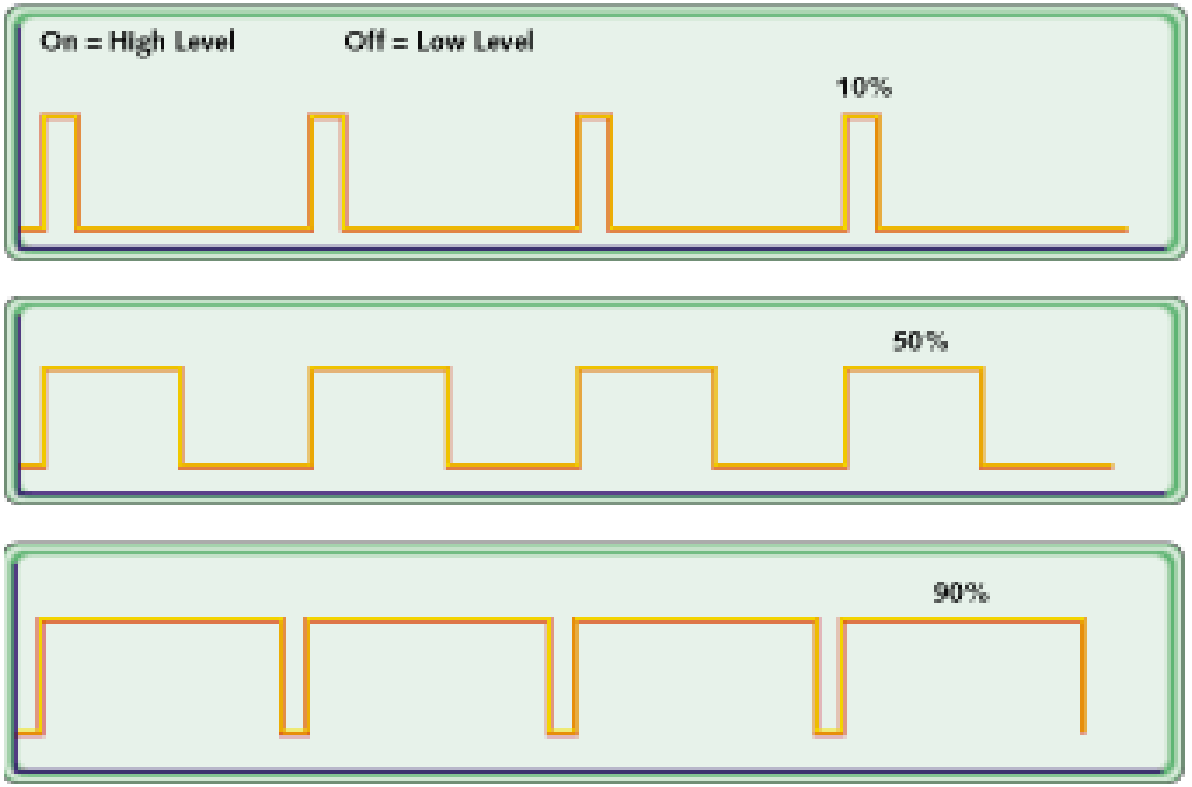
(RS = 0 control/RS =1 data)

- R/W’ = read or write port
- LCD controller has M display characters stored in ROM

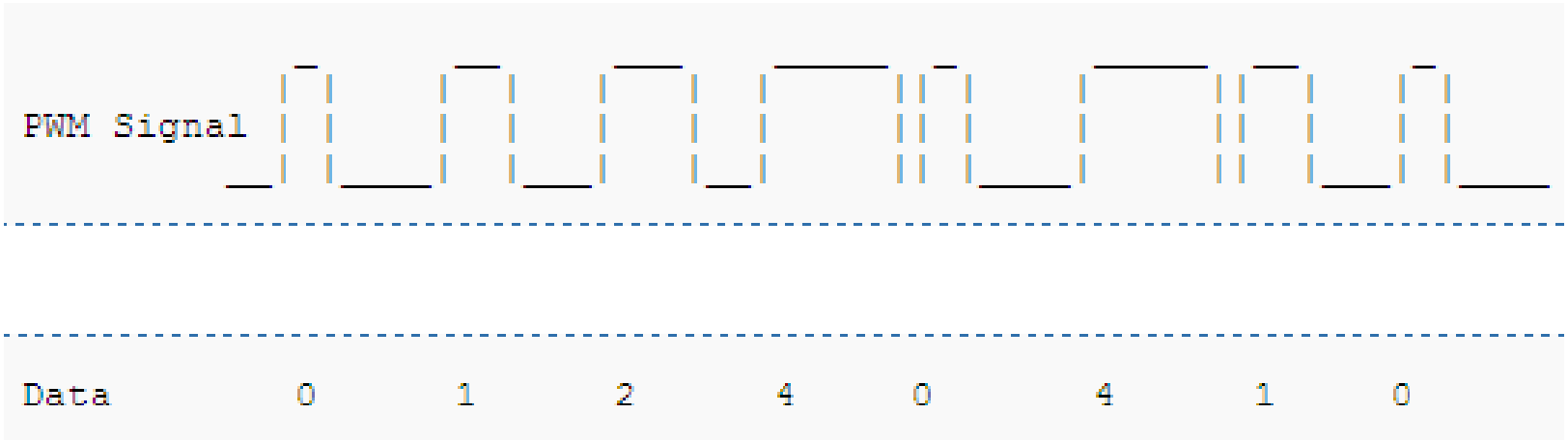
Pulse Width Modulation (PWM)

- PWM is a way for controlling analog circuits with digital output
- By controlling analog circuit digitally, system cost and power consumption can be reduced
- Used in audio, motor control, telecommunication
- The duty cycle is the proportional of the on time to the regular interval or period

PWM with 10%, 50%, and 90% duty cycle



PWM Signal



PWM Applications

- LED light control
- Motor speed
- Acoustics sound

PWM in RaspberryPi

Two software modules are available:

- RPI.GPIO is http://sourceforge.net/projects/raspberry-gpio-python
- RPIO is available from http:///pythonhosted.org/RPIO

GPIO Function in RPIO

Function	Description
`setmode(int num_mode)`	Identifies the method of pin numbering
`setup(int pin, int mode)`	Configures a pin as an input or output
`setup(int pin, int mode, int res_mode)`	Configures a pin as an input or output, and connects a pull-up or pull-down resistor
`output(int pin, int level)`	Sets a pin's logic level to `RPIO.HIGH` or `RPIO.LOW`
`int input(int pin)`	Reads the logic level at the given pin
`cleanup()`	Sets pins to default state
`add_interrupt_callback (int pin, callback_func, edge='both', pull_up_down=RPIO.PUD_OFF, threaded_callback=False debounce_timeout_ms=None)`	Associates a callback function with the given pin and an event that meets the specified criteria
`wait_for_interrupts (threaded=False, poll_timeout=1)`	Halts processing until an interrupt occurs
`del_interrupt_callback (int pin)`	Used to remove callbacks associated with the pin

RPIO Code Example:

```
import RPIO
# Set input pins
in_pin = 17;
out_pin = 24;
# Specify use of BCM pin numbering
RPIO.setmode(RPIO.BCM)

# Configure pin directions
RPIO.setup(in_pin, RPIO.IN)
RPIO.setup(out_pin, RPIO.OUT)

# Wait for in_pin to reach low voltage
while(RPIO.input(in_pin) == RPIO.LOW):
  RPIO.output(out_pin, RPIO.HIGH)

# Return pins to default state
RPIO.cleanup()
```

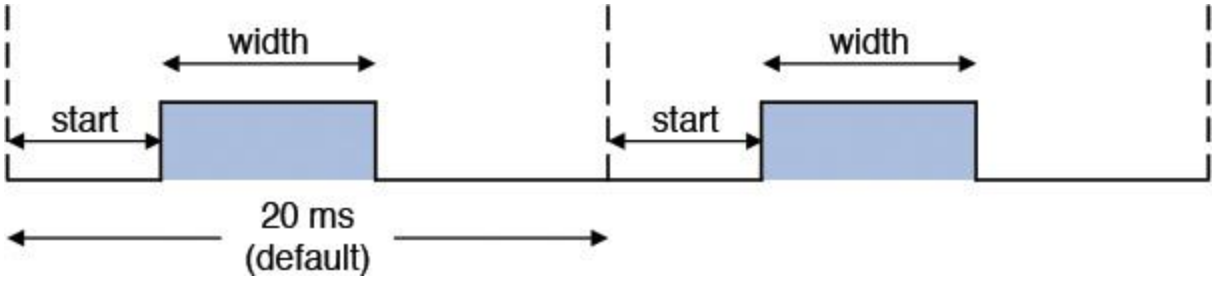
RPIO Code with Interrupt

```
import RPIO
def edge_detector(pin_num, rising_edge):
  if rising_edge:
    print("Rising edge detected on Pin %s" % pin_num)
  else:
    print("Falling edge detected on Pin %s" % pin_num)


in_pin = 17
RPIO.setmode(RPIO.BCM)
RPIO.setup(in_pin, RPIO.IN)


# Configure interrupt handling for rising and falling edges
RPIO.add_interrupt_callback(in_pin, edge_detector, edge='both')
RPIO.wait_for_interrupts()

RPIO.del_interrupt_callback(in_pin)
RPIO.cleanup
```

RPIO.PWM Module

Function	Description
`setup(pulse_incr_us=10, delay_hw=0)`	Initializes the DMA channels for use
`init_channel(int dma_channel, subcycle_time_us=20000)`	Configures the cycle with a specific period (20 ms by default)
`add_channel_pulse (int dma_channel, int pin, int start, int width)`	Generates pulse of the given width for the pin
`clear_channel (int dma_channel)`	Clears all pulses from the channel
`clear_channel_gpio (int dma_channel, int pin)`	Clears the pulse for the specified pin from the channel
`cleanup()`	Halts PWM and DMA

PWM Control with RPIO



RPIO Code with PWM

```
import RPIO.PWM as PWM
import time

# Define PWM pin
pwm_pin = 18

# Initialize DMA and set pulse width resolution
PWM.setup(1)

# Initialize DMA channel 0
PWM.init_channel(0)

# Set pulse width to 1000us = 1ms
PWM.add_channel_pulse(0, pwm_pin, 0, 1000)

time.sleep(10)

# Clear DMA channel and return pins to default settings
PWM.clear_channel(0)
PWM.cleanup()
```

RPIO Servo motor

```
import RPIO.PWM as PWM
import time


servo_pin = 18
min_width = 700
max_width = 2300

# Create servo object
servo = PWM.Servo()
# Set the angle to the minimum angle and wait
servo.set_servo(servo_pin, min_width)
time.sleep(1)
# Rotate shaft to maximum angle
for angle in xrange(min_width, max_width, 100):
  servo.set_servo(servo_pin, angle)
  time.sleep(0.25)
```

RPIO Servo motor

```
# Rotate shaft to minimum angle
for angle in xrange(max_width, min_width, -100):
  servo.set_servo(servo_pin, angle)
  time.sleep(0.5)

# Stop delivering PWM to servo
servo.stop_servo(servo_pin
```

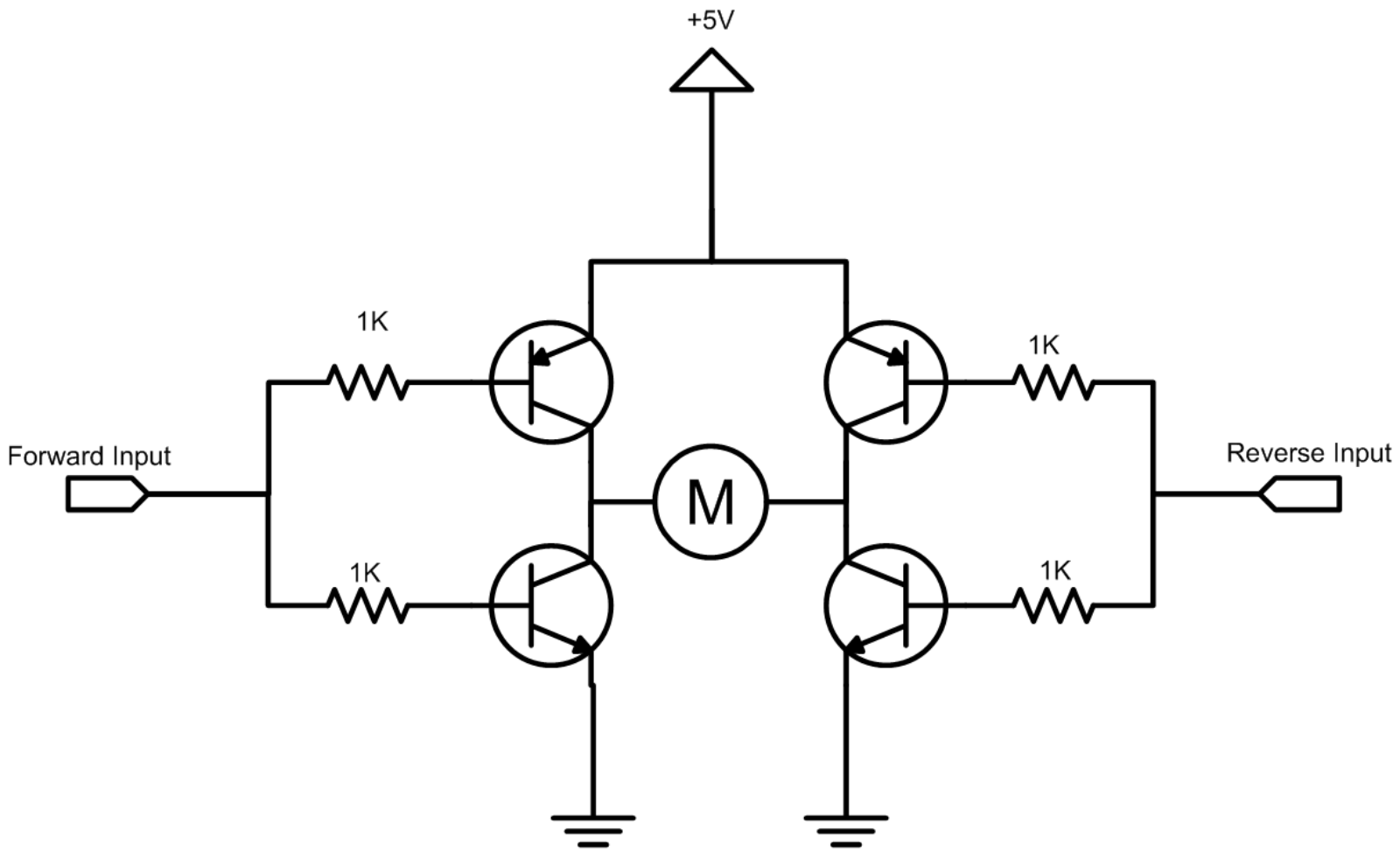
Driving Motors and Relays

- High current devices like motors, relays, solenoids, buzzers, and light bulbs can require more than 500mA of current
- Even though voltage levels may be the same, digital outputs from a GPIO (parallel) port typically drive only 5-20mA of current
- They cannot drive high current devices directly and trying to do so will likely **blow out the output circuit**

Driver Circuits

- A higher current driver circuit must be added after the digital output pin and before the device
- A driver circuit typically uses a discrete power transistor
- For DC motors, consider using an H-bridge circuit module. It contains four power transistors than can also reverse the motor.
- Diodes are often used for additional protection across the load on motors and relays. When you turn off the current in an inductive load it generates a reverse voltage spike that might damage the transistor (back EMF). The diode shorts it out.

H-Bridge - DC Motor Driver Circuit

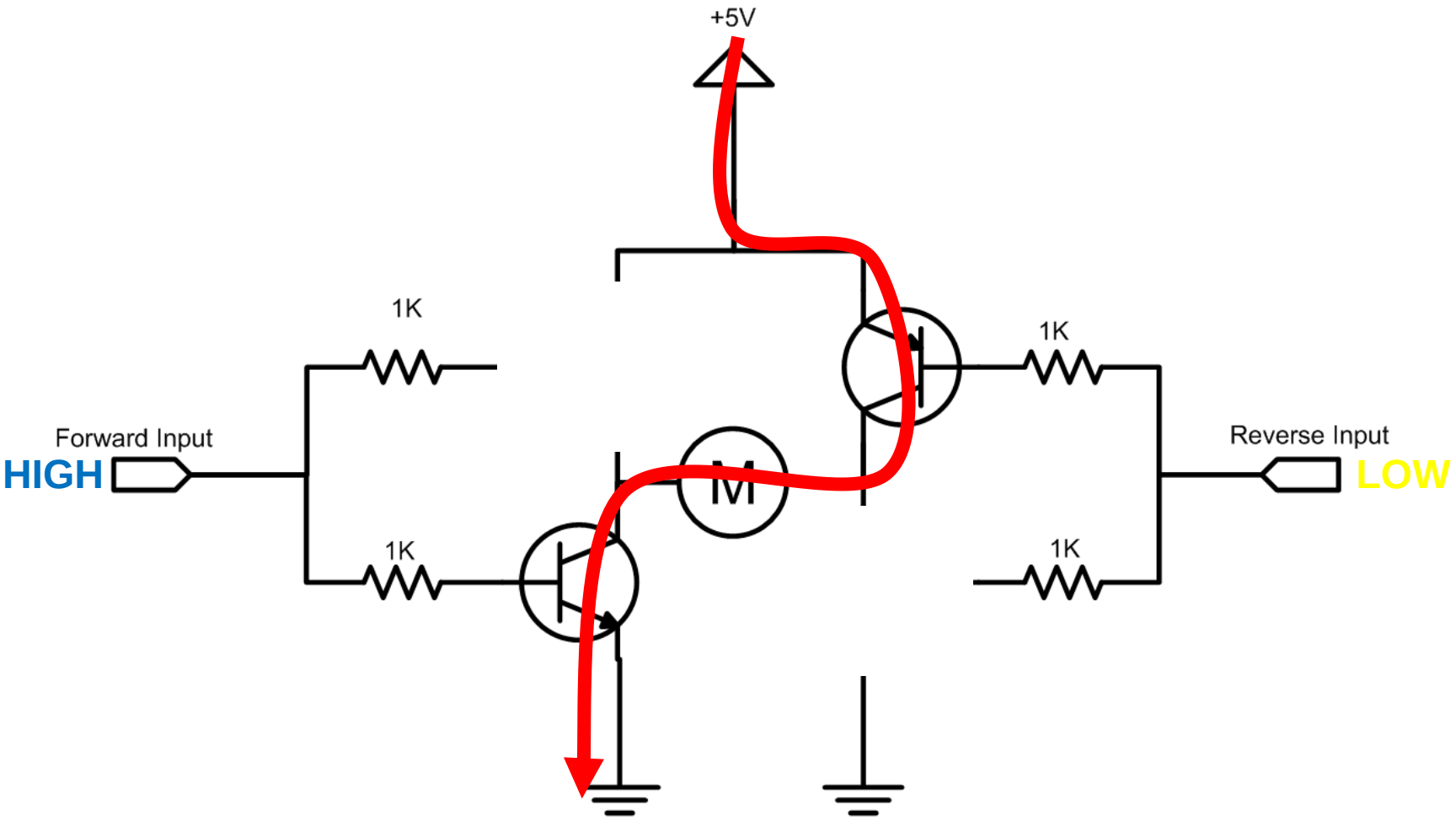


H-Bridge Control Functions

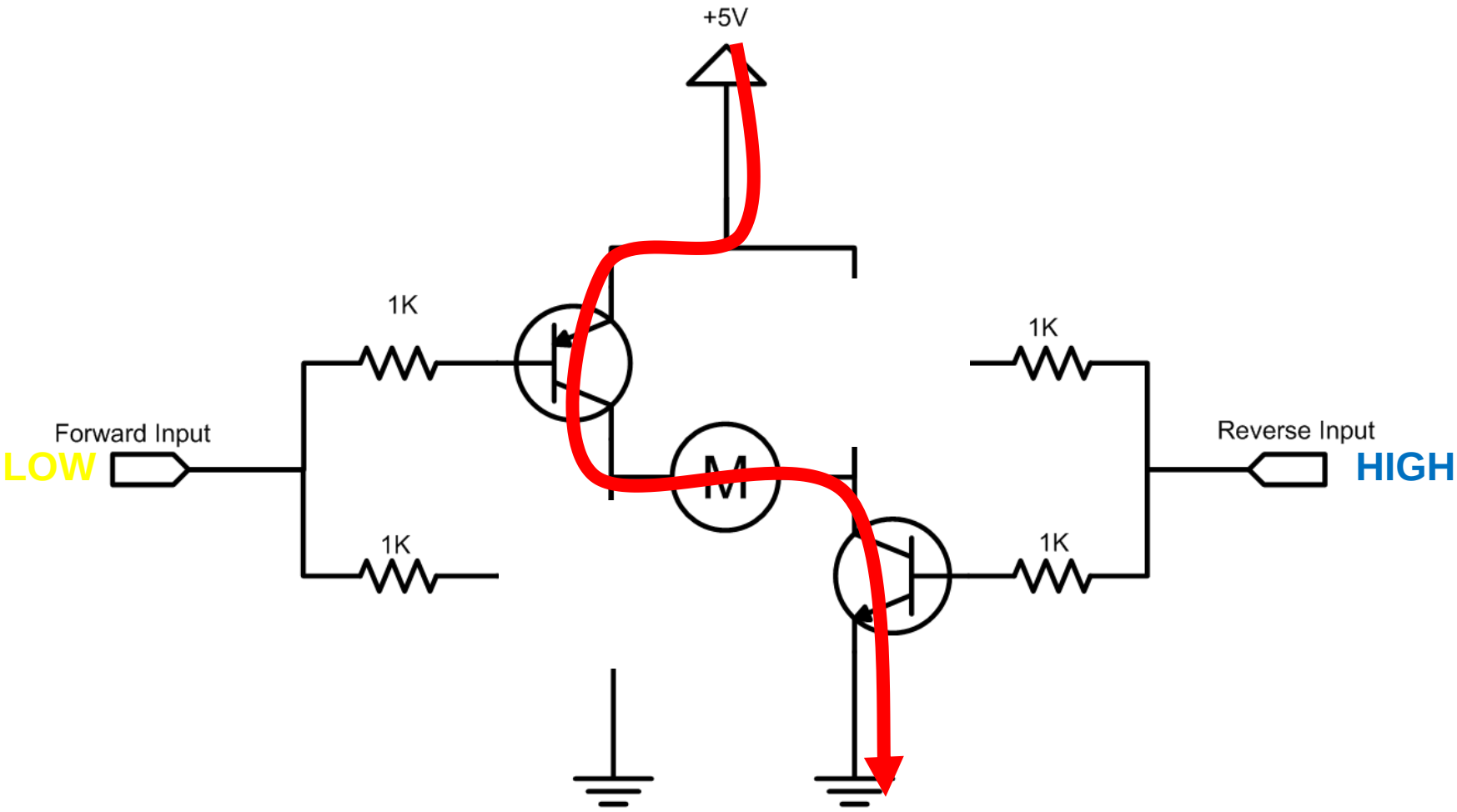
Input	Function	Operation
1 0	**Forward**	DC Motor runs in the forward direction
0 1	**Reverse**	DC Motor runs in the reverse direction
0 0	**Stop**	Motor is not connected – Coasts
1 1	**Brake* or** **Short Power Supply (not allowed!)**	Motor Terminals Shorted or Power Supply Shorted!

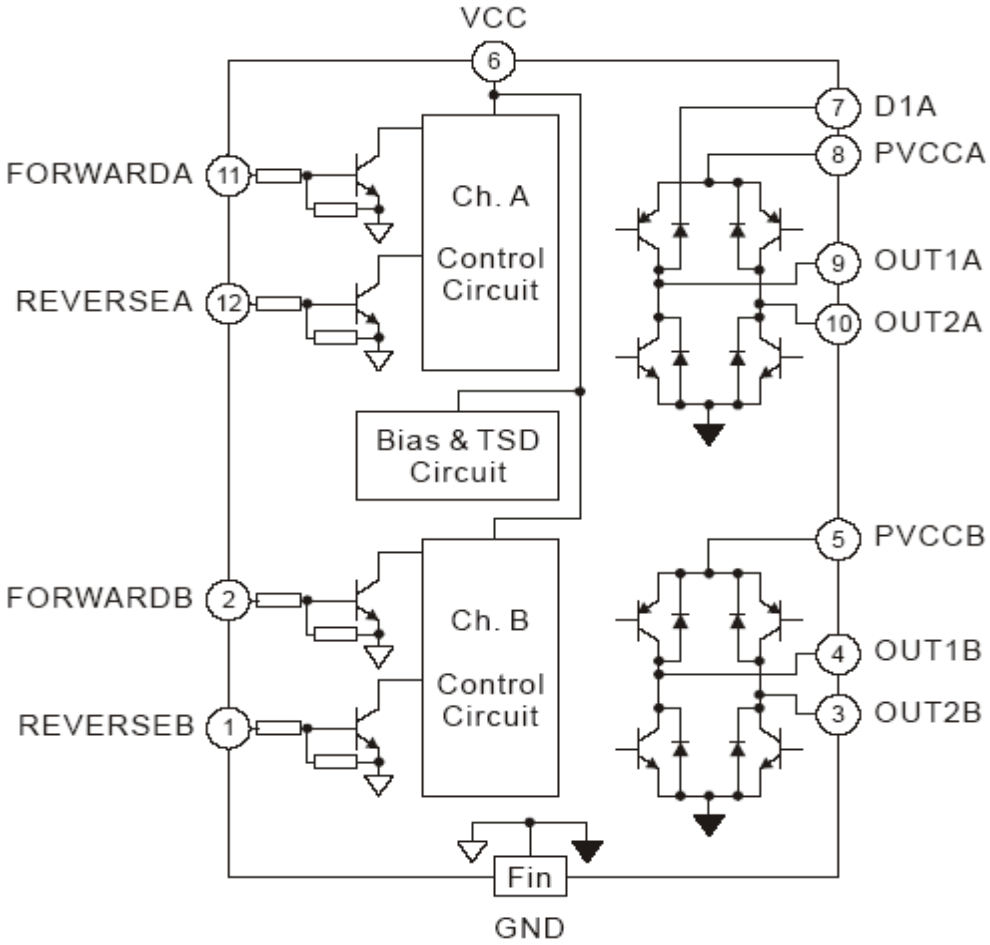
*The Brake function requires a more complex decoder circuit to control the power transistors. Check the H-Bridge data sheet to make sure it is supported before using it. In some simple H-Bridge circuits, the fourth state must be avoided (i.e., illegal state) and it will short out the power supply!

H-Bridge Example - Forward



H-Bridge Example - Reverse





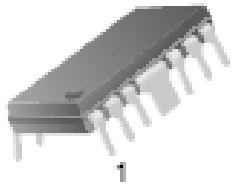
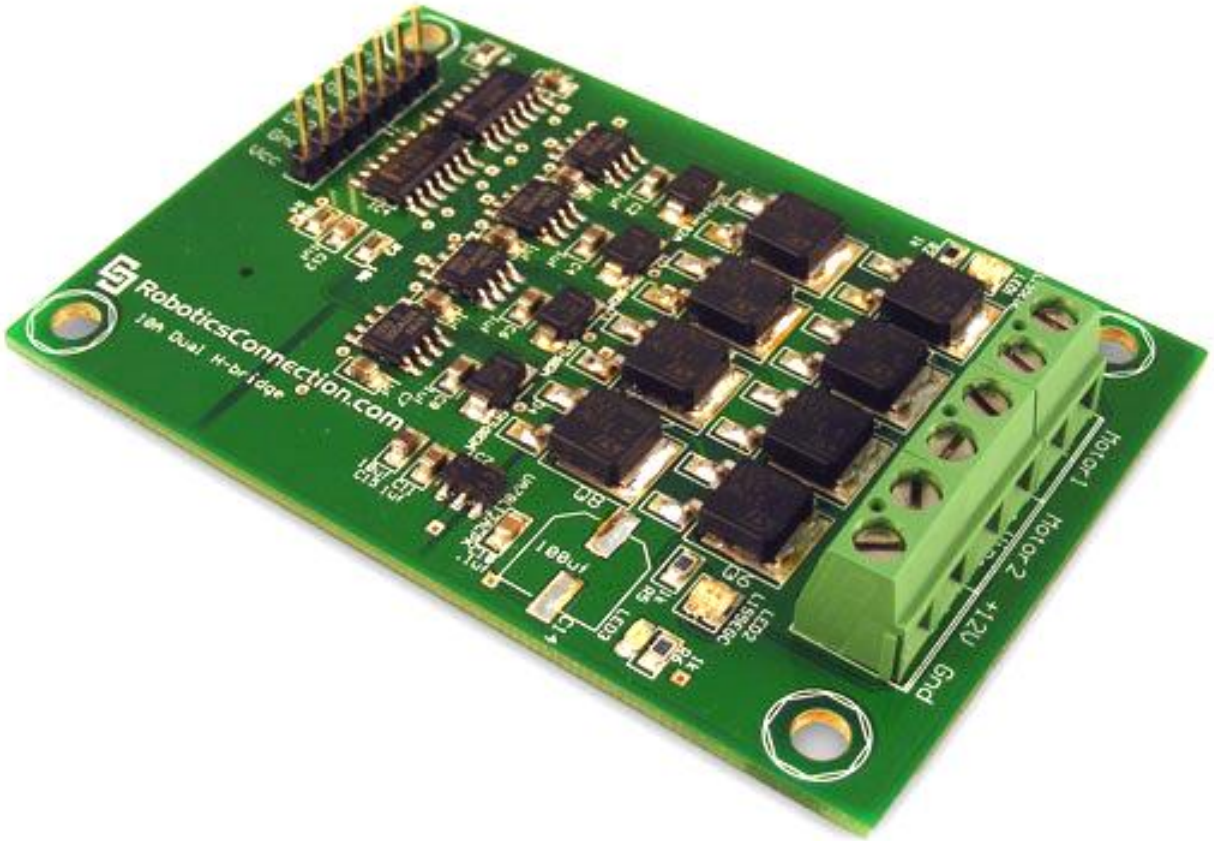


Figure 3.10 Fairchild FAN8100N Low Voltage Dual H-Bridge DC Motor Driver IC. Images courtesy of Fairchild Semiconductor.

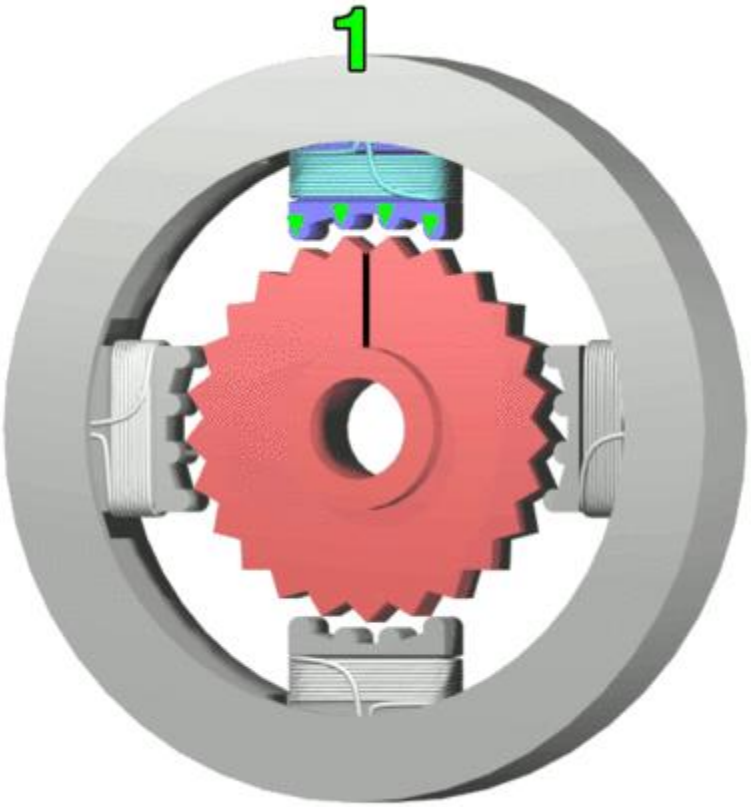


Higher current H-Bridge modules typically use discrete power transistors assembled on a board. This dual H-Bridge module switches up to 10 amps at 24V DC. The eight power transistors can be see arranged on the right side of the board. Photograph courtesy of RoboticsConnection.

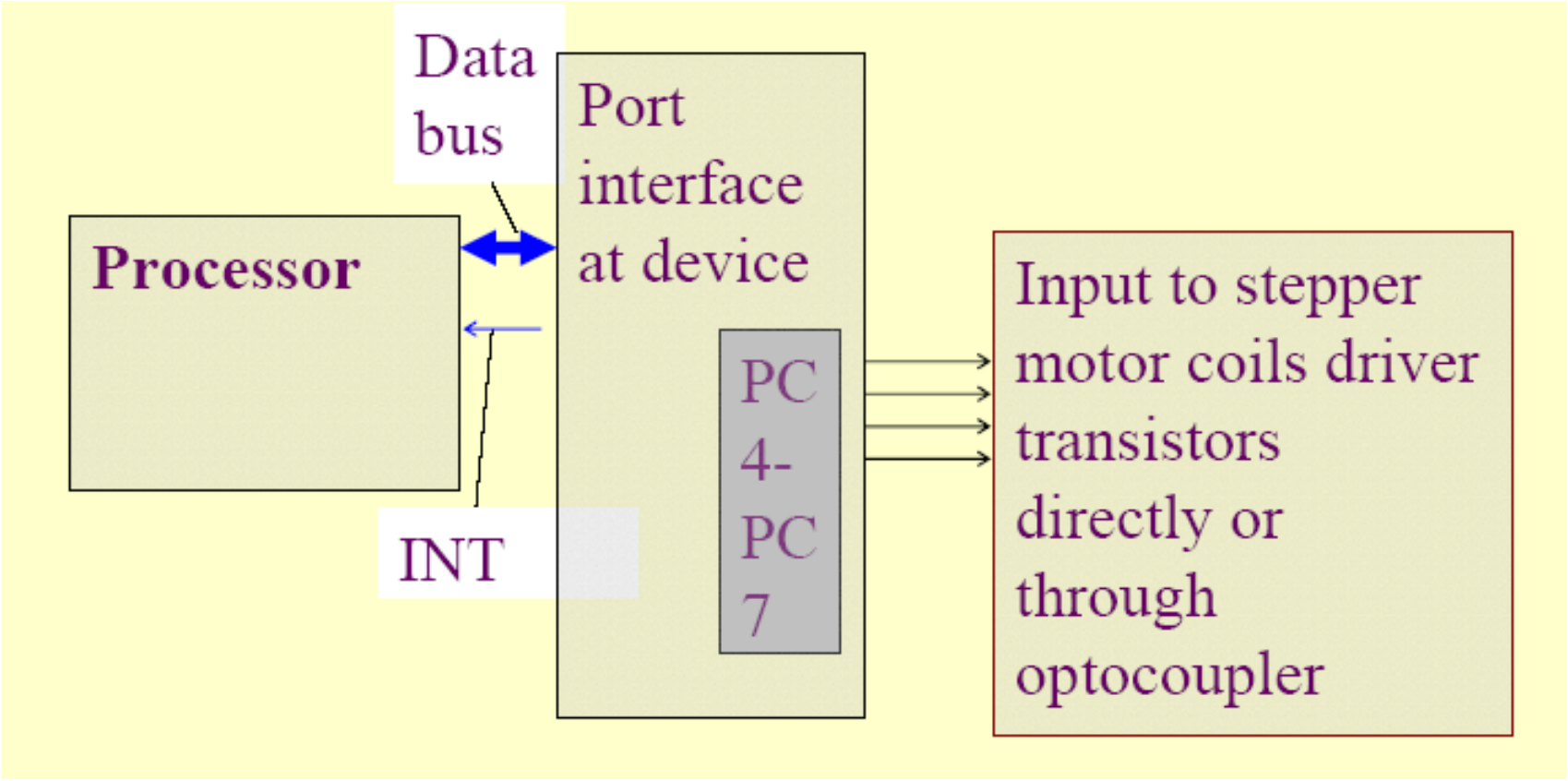
Stepper motor

- A synchronous electric motor that can divide a full rotation into a number of steps
- Motor position can be controlled precisely without any feedback system
- Doesn’t require feedback sensor
- Operate in DC power
- Used in many devices such as harddisk drives, and printers
- Can make motor spin by outputting the sequence like … 10,9,5,6,10,9,5,6….
- For 200 steps motor, each new output will cause the motor to rotate 1.8 degree

Stepper motors



Stepper motor



Raspberry pi Camera

Raspberry Pi can interface through:

- Camera Serial Interface (CSI)
- USB
- IP Camera

RasPi Camera modules

- Raspberry Pi camera module: 5 M pixels at 30 FPS

- Raspberry Pi camera module black NoIR

OpenCV Installation

https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

sudo raspi-config

- enable SSH
- enable Camera

sudo apt-get update

sudo apt-get upgrade

sudo apt-get –y install synaptic

OpenCV Installation

sudo apt-get -y install python-numpy python-scipy python-nose python-pandas python-matplotlib ipython-notebook python-sympy

sudo apt-get -y install libgtkglext1-dev

sudo apt-get -y install build-essential cmake pkg-config

sudo apt-get -y install qtcreator qt4-dev-tools libqt4-dev libqt4-core libqt4-gui v4l-utils

OpenCV Installation

wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.10/opencv-2.4.10.zip

unzip opencv-2.4.10.zip

tar xzvf opencv-2.4.10.tar.gz

cd opencv-2.4.10

mkdir build

cd build

OpenCV Installation

cmake -D CMAKE_BUILD_TYPE=RELEASE -D WITH_OPENGL=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D WITH_QT=ON -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D WITH_V4L=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D BUILD_EXAMPLES=ON

make

sudo make install

sudo nano /etc/ld.so.conf.d/opencv.conf

    add /usr/local/ib

sudo nano /etc/bash.bashrc

    add

PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig

export PKG_CONFIG_PATH

Testing

lsusb # list all usb devices

sudo apt-get –y install guvcview # capture image

lxsession

guvcview

cd /home/pi/opencv-2.4.10/samples/python

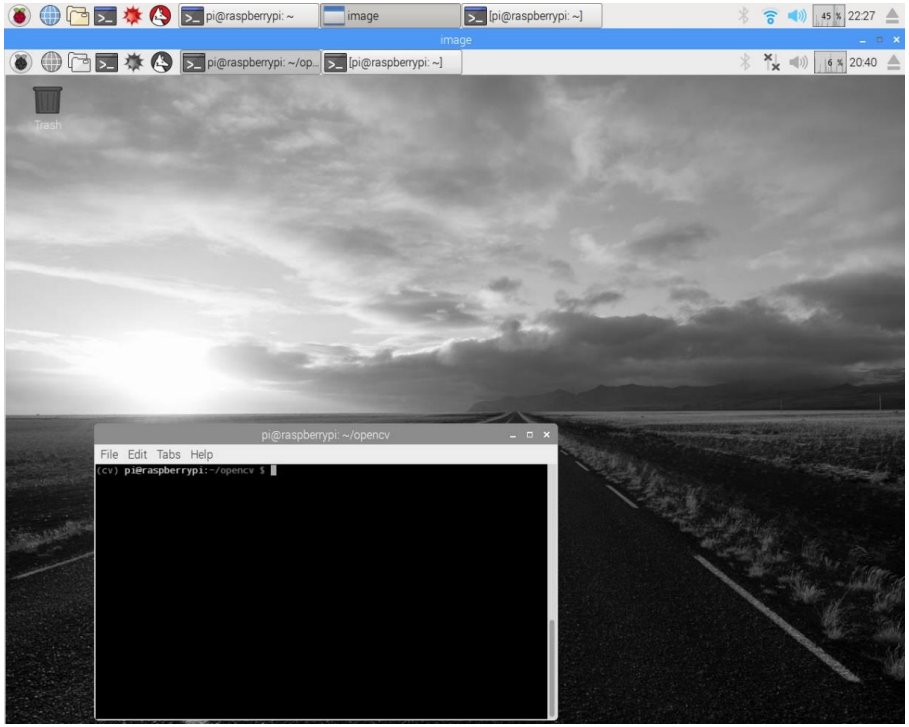
ls

python facedetect.py

OpenCV Example

```
import cv2

img = cv2.imread('/home/pi/screenshot.jpg',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Result


pi@raspberrypi: ~
image
[pi@raspberrypi: ~]
22:27
image
pi@raspberrypi: ~/op...
[pi@raspberrypi: ~]
20:40
Trash
pi@raspberrypi: ~/opencv
File Edit Tabs Help
(cv) pi@raspberrypi:~/opencv $

Bus history

- Industry System Architecture (ISA)
- Developed by IBM in 1981 for IBM XT (extended technology) 8 bits
- Modified for 16 bits for IBM AT (advance technology)
- Later, they rename AT bus to ISA bus
- IBM move to replace IBM AT bus with MCA (Micro Channel Architecture)

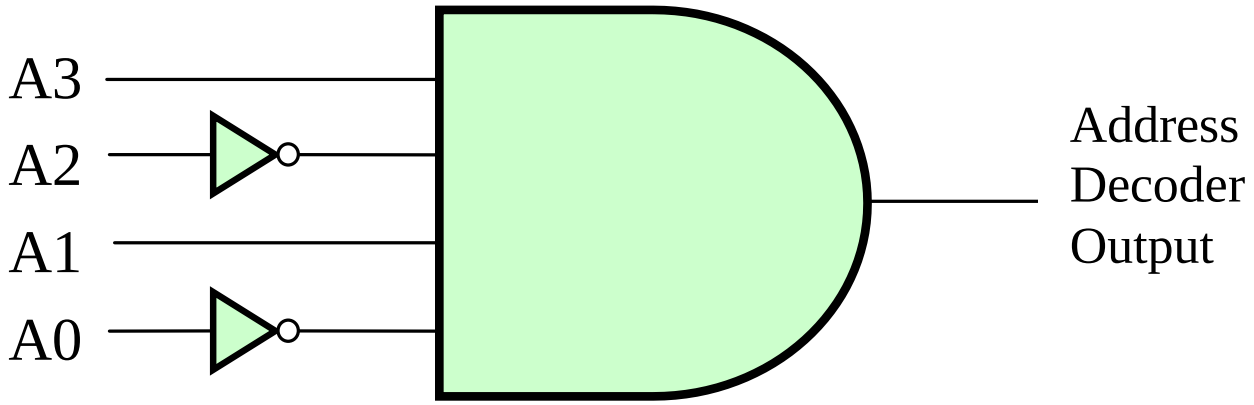
Bus history

- Gang of nine IBM PC compatible manufacturers announces EISA to fight with MCA for 32 bits
- PCI (Peripheral Component Interface) is developed by Intel in 1990 (my ex-manager was in that team)
- AGP Accelerated Graphics Port
- PCI-E (Express) is the extension of PCI
- ARM bus is designed for ARM architecture

A First Generation Bus Example: ISA

- ISA bus used in early PCs for Plug-in Cards
- Address Bus (SA*x*)
  - Originally 16 bits then expanded to 24
- Data Bus (SD*x*)
  - 8 bits then 16 bits
  - EISA expanded it to 32 bits
- Bus Status Signals
  - MEMR, MEMW, IOR, IOW
- 5V TTL signals

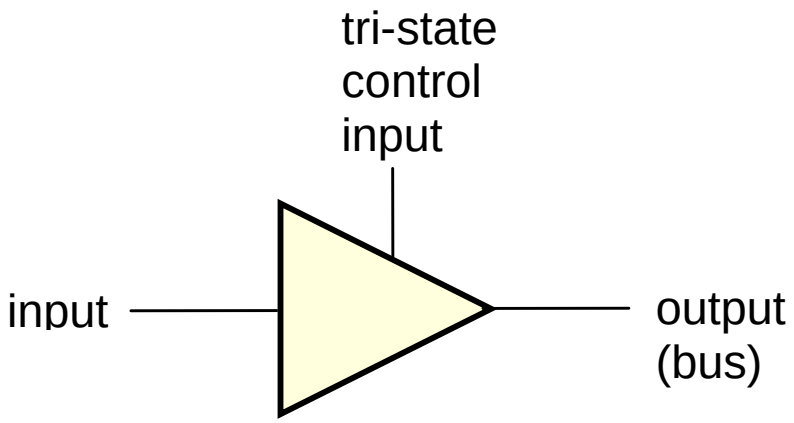
Digital Logic Review:
A simple address decoder circuit for 4-bit address (A3..A0) = 0xA = 1010B



Would need to decode more address bits in an actual system

Tri-state logic gate outputs are used to drive most bus signals

control	input	output
0	0	High Z
0	1	High Z
1	0	0
1	1	1



Hardware allows only one tri-state gate at a time to drive a bus signal!

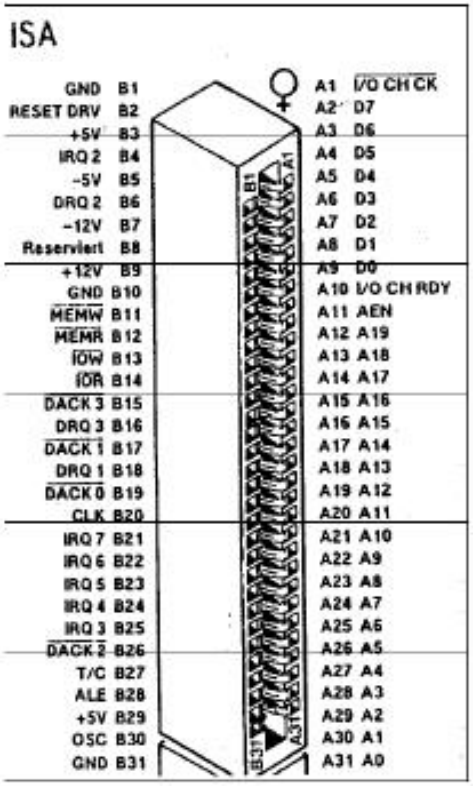
Works just like a large multiplexer:
One of several inputs connects to the output

Legacy PC I/O address assignments

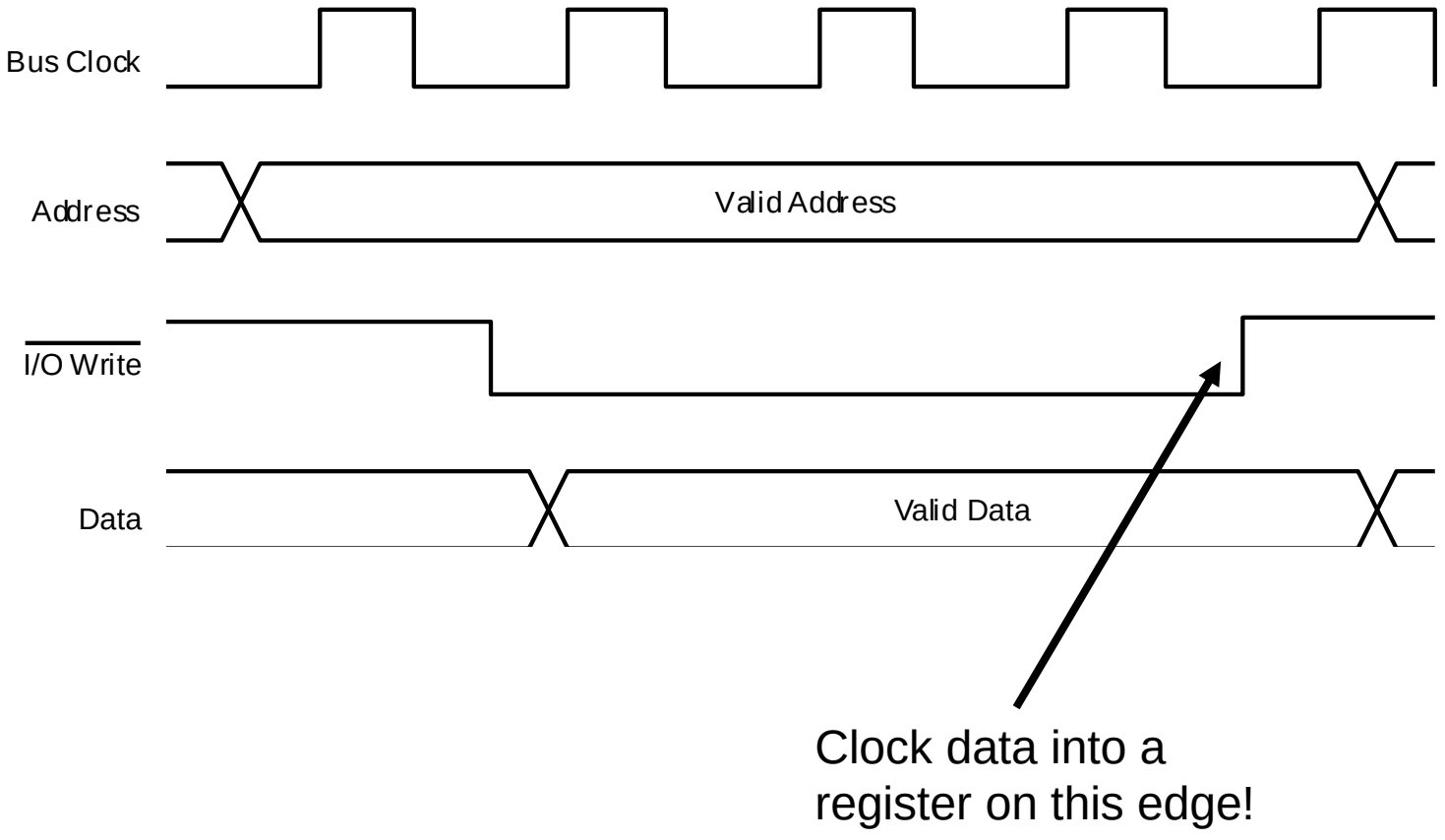
I/0 address range	I/O device
000h – 200h	Reserved for Internal Devices: Interrupt & DMA controllers, timers
278h - 27Fh	Parallel Printer (LPTx:)
2E8h - 2EFh	Serial Port 4 (COM4:)
2F8h - 2FFh	Serial Port 2 (COM2:)
378h - 37Fh	Parallel Printer (LPT1:)
3B0h - 3BBh	MDA Adapter
3BCh - 3BFh	Parallel Printer (LPTx:)
3C0h - 3CFh	VGA/EGA Adapter
3D0h - 3DFh	CGA Adapter
3E8h - 3EFh	Serial Port 3 (COM3:)
3F0h - 3F7h	Floppy Controller
3F8h - 3FFh	Serial Port 1 (COM1:)

Original PC design only decoded low 10 I/O address bits to save hardware

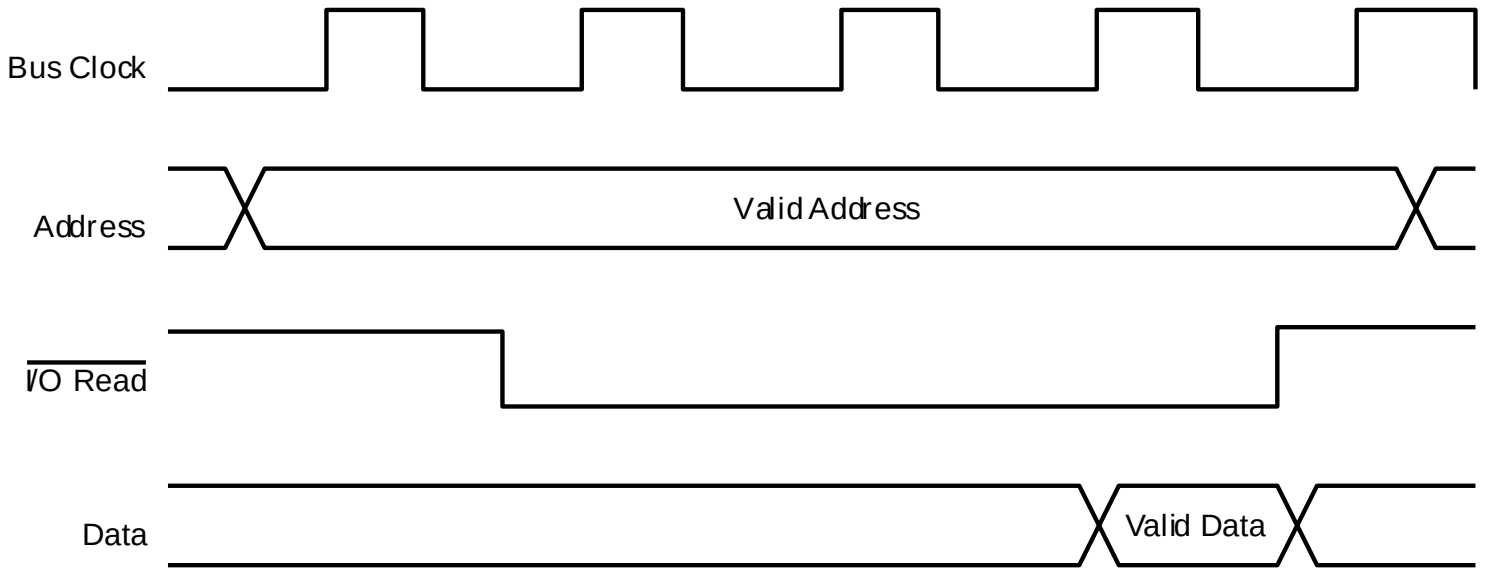
ISA bus


ISA
GND B1
RESET DRV B2
+5V B3
IRQ 2 B4
-5V B5
DRQ 2 B6
-12V B7
Reserviert B8
+12V B9
GND B10
MEMW B11
MEMR B12
IOW B13
IOR B14
DACK 3 B15
DRQ 3 B16
DACK 1 B17
DRQ 1 B18
DACK 0 B19
CLK B20
IRQ 7 B21
IRQ 6 B22
IRQ 5 B23
IRQ 4 B24
IRQ 3 B25
DACK 2 B26
T/C B27
ALE B28
+5V B29
OSC B30
GND B31
A1 I/O CH CK
A2 D7
A3 D6
A4 D5
A5 D4
A6 D3
A7 D2
A8 D1
A9 D0
A10 I/O CH RDY
A11 AEN
A12 A19
A13 A18
A14 A17
A15 A16
A16 A15
A17 A14
A18 A13
A19 A12
A20 A11
A21 A10
A22 A9
A23 A8
A24 A7
A25 A6
A26 A5
A27 A4
A28 A3
A29 A2
A30 A1
A31 A0
B1
A1
B31
A31

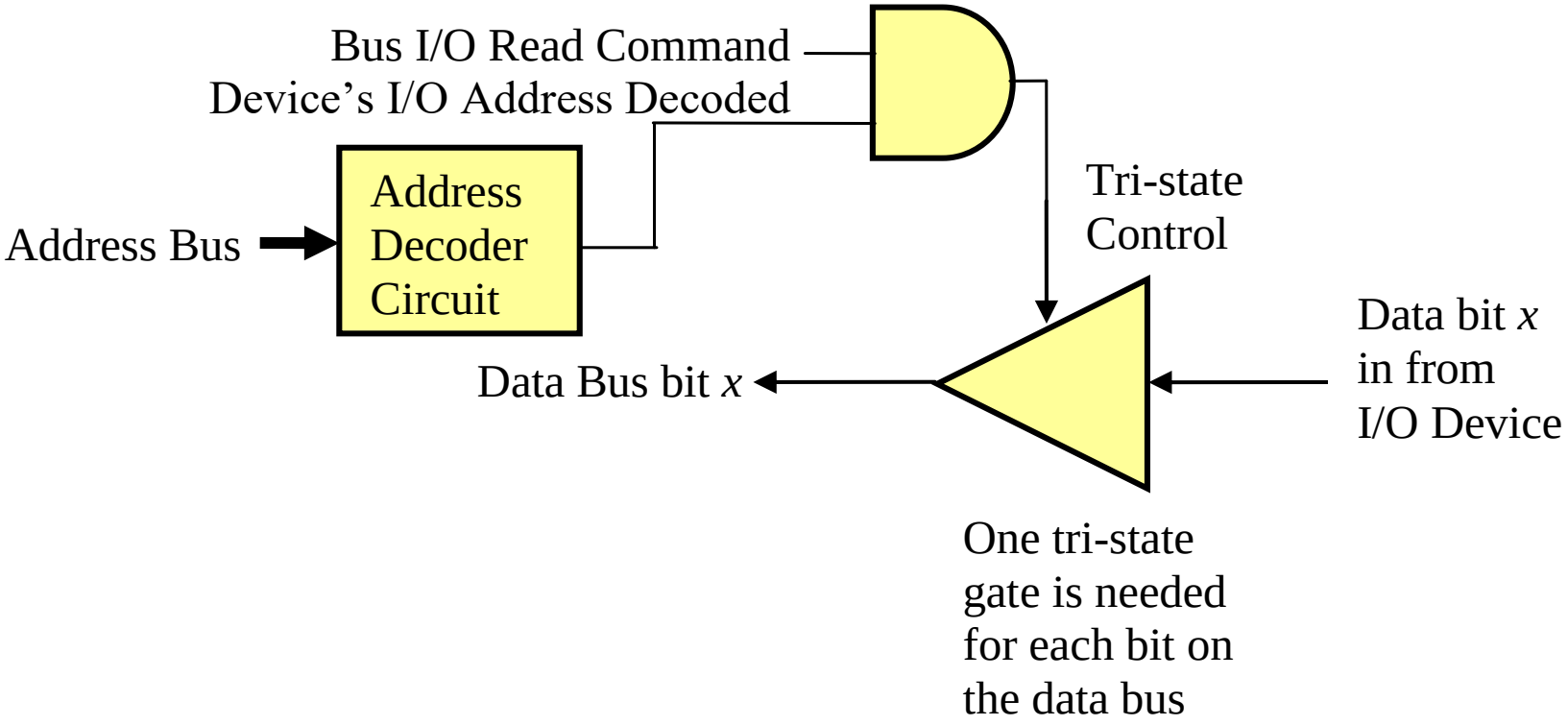
An Example ISA Bus I/O Write Operation



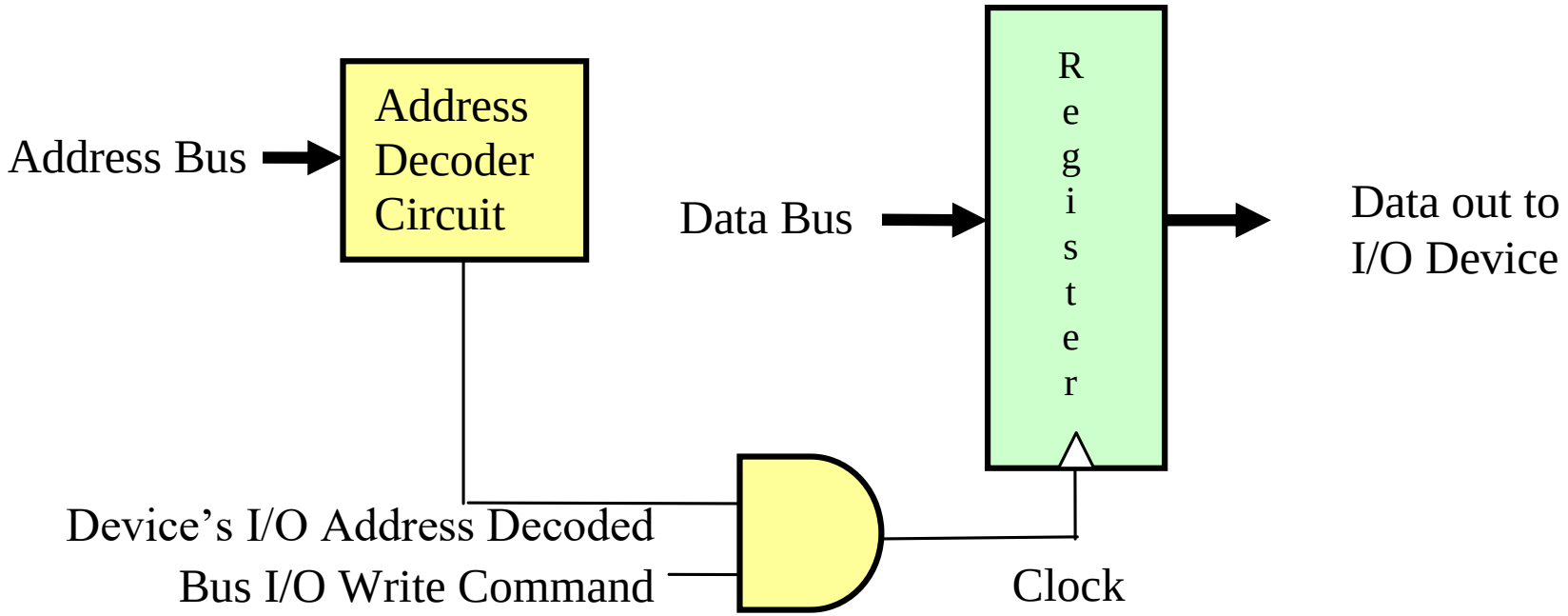
An Example ISA Bus I/O Read Operation



Typical I/O Input Port Hardware Operation



Typical I/O Output Port Hardware Operation



Software for I/O port transfers

- Can use in-line assembly language in C/C++
- Most C/C++ compilers have built-in function calls for I/O port input and output

In-line Assembly Example for X86

```
// I/O Input Routine
__asm{
   mov   dx, IO_address
   in    al, dx
   mov   IO_data,al
}
```

```
// I/O Output Routine
__asm{
   mov  dx,IO_address
   mov  al,IO_data
   out        dx, al
}
```

Problems: Does not port to other processors and many people do not understand assembly language!

Windows CE C/C++ I/O Port R/W Functions

- **READ_PORT_UCHAR**(*I/O_Address*)
  - Returns 8-bit input value from port
- **WRITE_PORT_UCHAR**(*I/O_Address*,*I/O_Data*)
  - Sends 8-bit data value to output port
- Found in CE Device Driver Kit (CEDDK)
- Need to link to CEDDK.lib library and include CEDDK.h – modify *sources* file for this
- Typically used in low-level device drivers

A Second Generation Bus - PCI

- 32-bit Multiplexed Address and Data Bus (AD)
- Address sent on first clock cycle
- Bus command sent on first clock cycle (C/BE)
- Data on subsequent clock cycles
- Bus Clock rates 33 to 512Mhz
- One Data transfer per clock is possible
- Supports Data Bursts (example to follow)

Computer System PCI

- When I/O devices are distributed in the network, all can communicate through a common parallel bus
- PCI connects at high speed to other devices on a very short distances (<25 cm) using a parallel bus

PCI bus applications

- Display monitor
- Printer
- Streaming displays
- Network subsystems
- Video card
- Digital video capture card
- Harddisk controller

PCI bus features

- 32 bit data bus extendible to 64 bit
- Support both Synchronous and Asynchronous transfer
- Automatically detects the interfacing systems and assign new address
- Simplify the addition and deletion of the system peripherals

Identification numbers

A device identifies its address space by three identification numbers

- I/O port
- Memory locations
- Configuration registers

Each PCI device has address space allocation of 256 bytes

PCI device

- A 32-bit register in a PCI device for device ID and allow auto-detection
- Each device may use a FIFO controller with a FIFO buffer for maximum throughput
- Independent from IBM architecture
- Number of embedded devices use PCI bus

PCI bridge

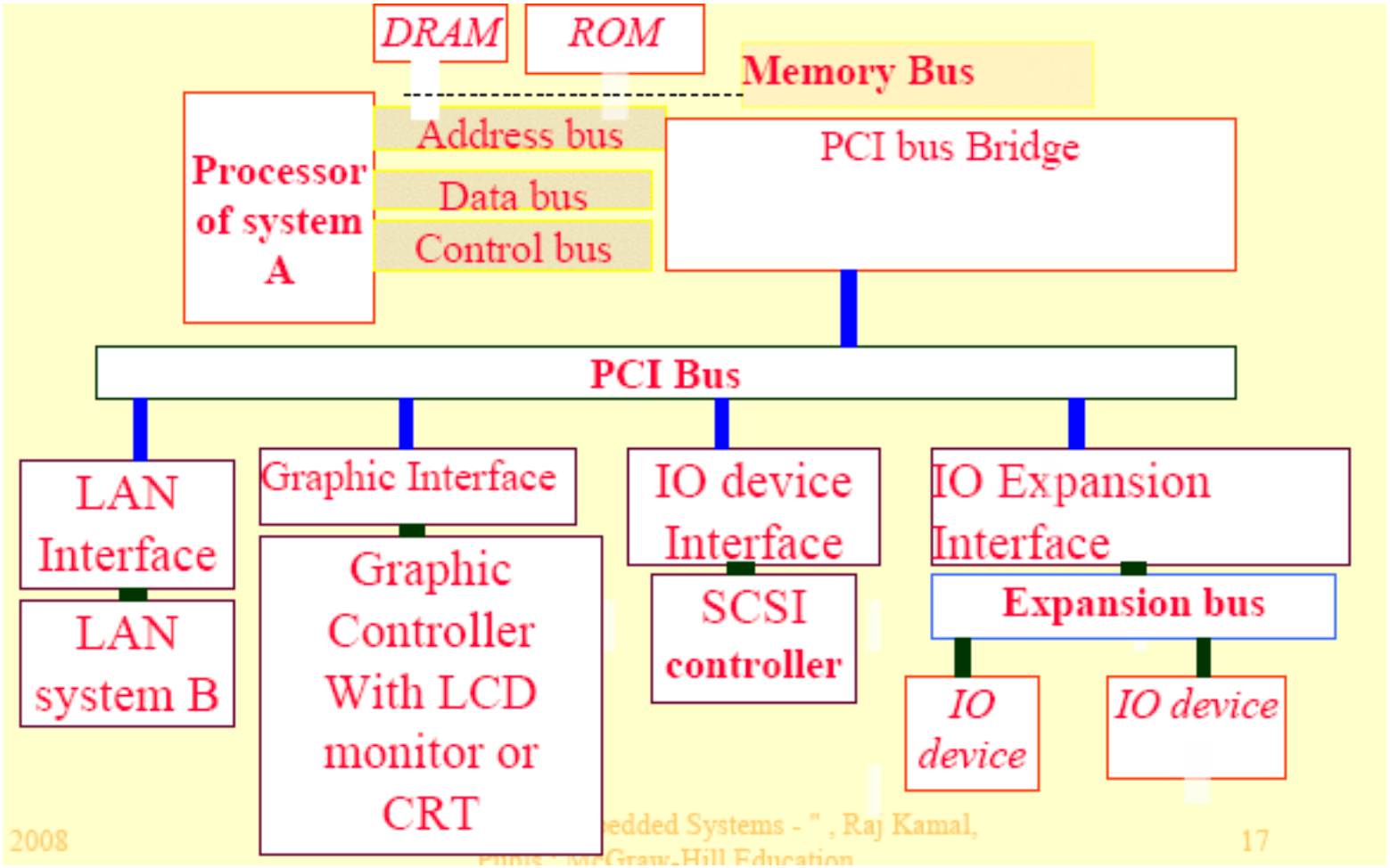
- PCI bridge switches the communication between memory bus to PCI bus
- In most systems, the processor has a single bus that connects to a PCI bridge
- Some processor integrates PCI bridge within the processor to reduce the system cost

PCI bridge/switch

Another configuration:

- Separate memory bus
- A separated I/O bus connected to PCI switch to the I/O devices
- Widely used in desktop PC

PCI bridge and buses



PCI

- 32 bit 33 MHz

  throughput = 133 MBps

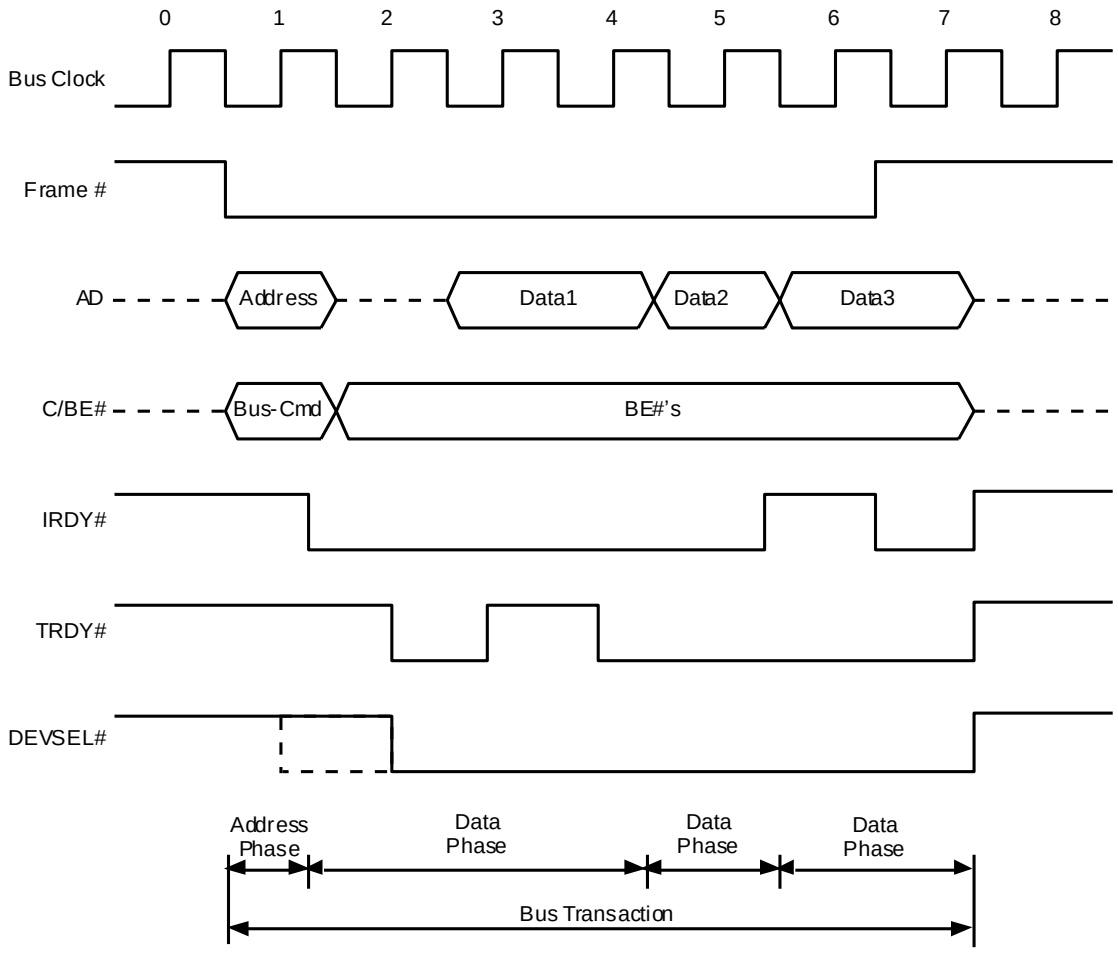
- 64 bit 66 Mhz

  throughput = 533 MBps

PCI Bus Commands (C/BE)

PCI Bus Command	C/BE
Interrupt Acknowledge	0000
Special Cycle	0001
I/O Read	0010
I/O Write	0011
Reserved	0100
Reserved	0101
Memory Read	0110
Memory Write	0111
Reserved	1000
Reserved	1001
Configuration Read	1010
Configuration Write	1011
Memory Read Multiple	1100
Dual Address Cycle	1101
Memory Read Line	1110
Memory Write and Invalidate	1111

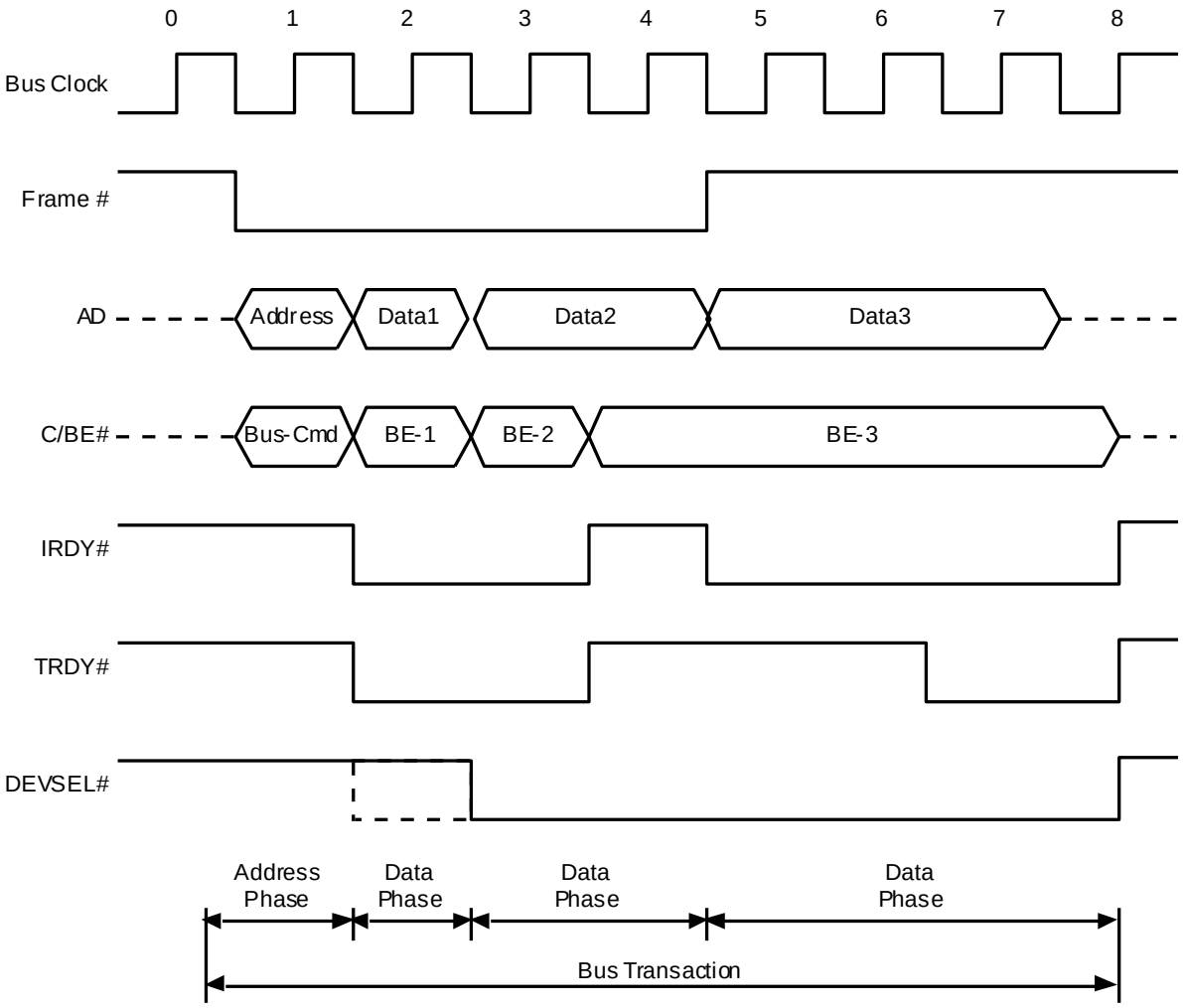
PCI Read Burst Cycle



PCI Read Burst Cycle

Clock Cycle	Description of PCI operation
0	Bus is idle
1	The initiator sets FRAME low, places the address on the Address/Data (AD*x*) lines, and the bus command (read) on the Command/Byte Enable (C/BE) lines (address phase).
2	The initiator tri-states the address and waits for the target to return a data value by turning on its tri-state drivers. Device Select (DEVSEL) low indicates a target device has decoded its address range and it is responding to the command. The target drives TRDY high to indicate the target needs another clock cycle to respond with the data.(data phase)
3	The target drives the data value and sets target ready (TRDY) low to indicate that data is valid. When both IRDY and TRDY are low a data transfer occurs.
4	The target sets TRDY high to indicate it need an additional clock cycle for the next data transfer.
5	The second data transfer occurs when both TRDY and IRDY are low. The initiator saves the target data.
6	The target drives the data value, but the initiator requests an additional clock cycle by set IRDY high.
7	The initiator sets IRDY low to complete the third data transfer. The initiator saves the target data value, The initiator drives FRAME high to end the data phase.
8	All bus signals are tri-stated or driven to the inactive state.

PCI Write Burst Cycle



PCI Write Burst Cycle

Clock Cycle	Description of PCI operation
0	Bus is idle
1	The initiator sets FRAME low, places the address on the Address/Data (AD*x*) lines, and the bus command (write) on the Command/Byte Enable (C/BE) lines (address phase).
2	The initiator places the data on the AD*x* lines and byte enables on C/BE lines, Device Select (DEVSEL) low indicates a target device has decoded it's address range and it is responding to the command. When both IRDY and TRDY are low the target saves the data. (data phase)
3	The initiator drives new data and byte enables. When both initiator ready IRDY and TRDY are low a data transfer occurs and the target saves the data.
4	The initiator sets IRDY high and the target sets TRDY requesting an additional clock cycle.
5	The initiator drives new data and byte enables and sets IRDY low. The initiator sets FRAME high indicating the final data transfer.
6	The target drives the data value, but the initiator requests an additional clock cycle by set IRDY high.
7	The initiator sets IRDY low to complete the third data transfer. The target saves the data value.
8	All bus signals are tri-stated or driven to the inactive state.

Software for PCI devices

- Each PCI device has a 256 byte configuration area
- At power up each device can respond with manufacturer and device type information
- Allows system to locate and load device drivers at power up
- Memory and I/O base addresses are configured with software (no jumpers)

Accelerated Graphics Port (AGP)

- Newer Graphics cards were consuming most of the PCI bus bandwidth
- Solution: Move graphics card to its own PCI bus
- A connection with only 1 device is technically a “port” and not a “bus.”
- Extra clock phase signals were added to increase clock rate 2X, 4X, and 8X

Logic Analyzer to capture PCI bus

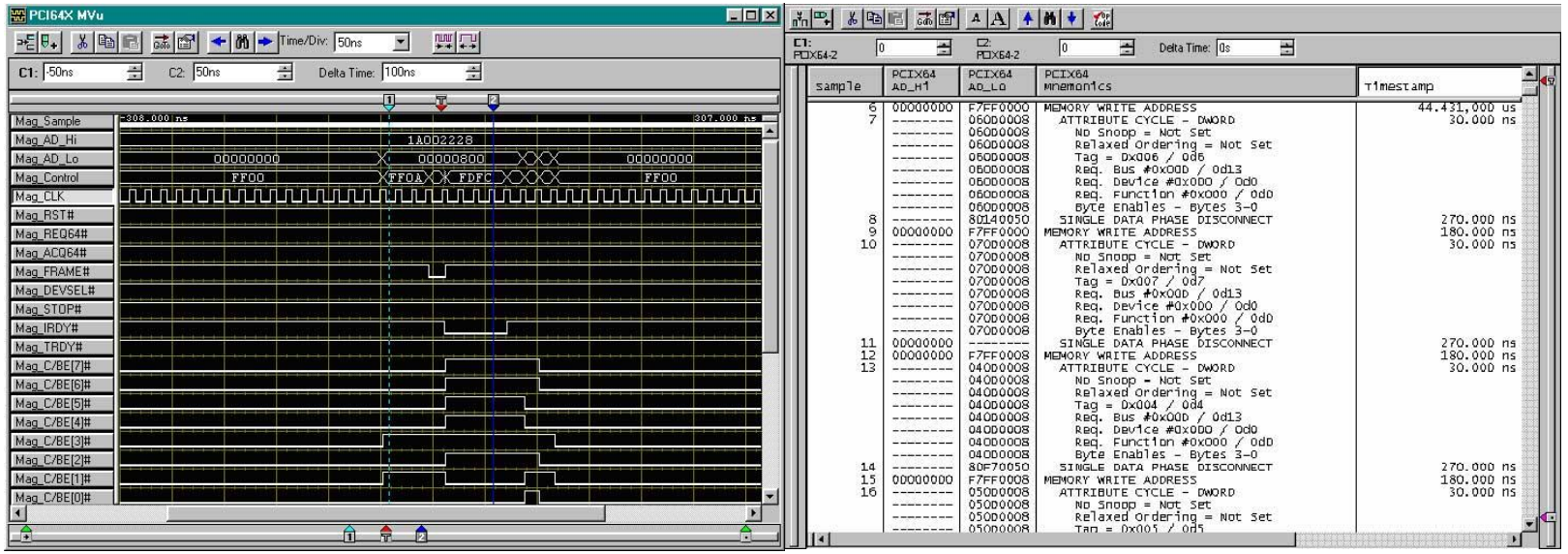


Figure 2.15 A Tektronix logic analyzer setup to capture and decode PCI bus signals. This logic analyzer is actually an embedded system that runs Windows XP. A special PCI interposer adapter card is used to quickly connect probes along with special firmware for the logic analyzer. Images courtesy of Nexus Technology.

Comparing ISA/PCI

	ISA	PCI
Adv:		
	16 bits	32 bits
	slower	faster
	separated addr/data	shared addr/data
	only PC system	support many systems
	simple protocol	PCI bridge

Third Generation Bus – PCI Express

- High-Speed Serial line(s) are used to transfer PCI signals
- Fewer signal lines are used, but with much higher bandwidth on each signal line
  - More stringent design restrictions on drivers, length, loading, crosstalk, and terminations
- Clock rates from 2.5 Gbps to 10 Gbps
- Can combine serial lines into groups called lanes to provide more bandwidth to a device
- No changes needed in software – works the same as PCI

PCI-E speed

A lane is composed of a transmit and receive pairs of differential lines (4 wires)

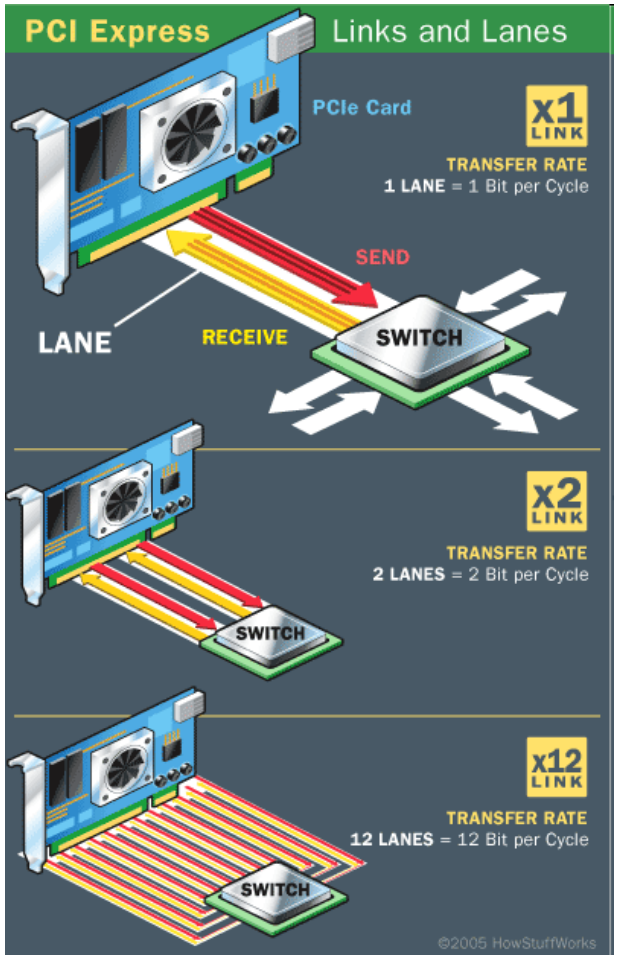
Per lane:

- **v1.x**: 250 MB/s
- **v2.0**: 500 MB/s
- **v3.0**: 1 GB/s

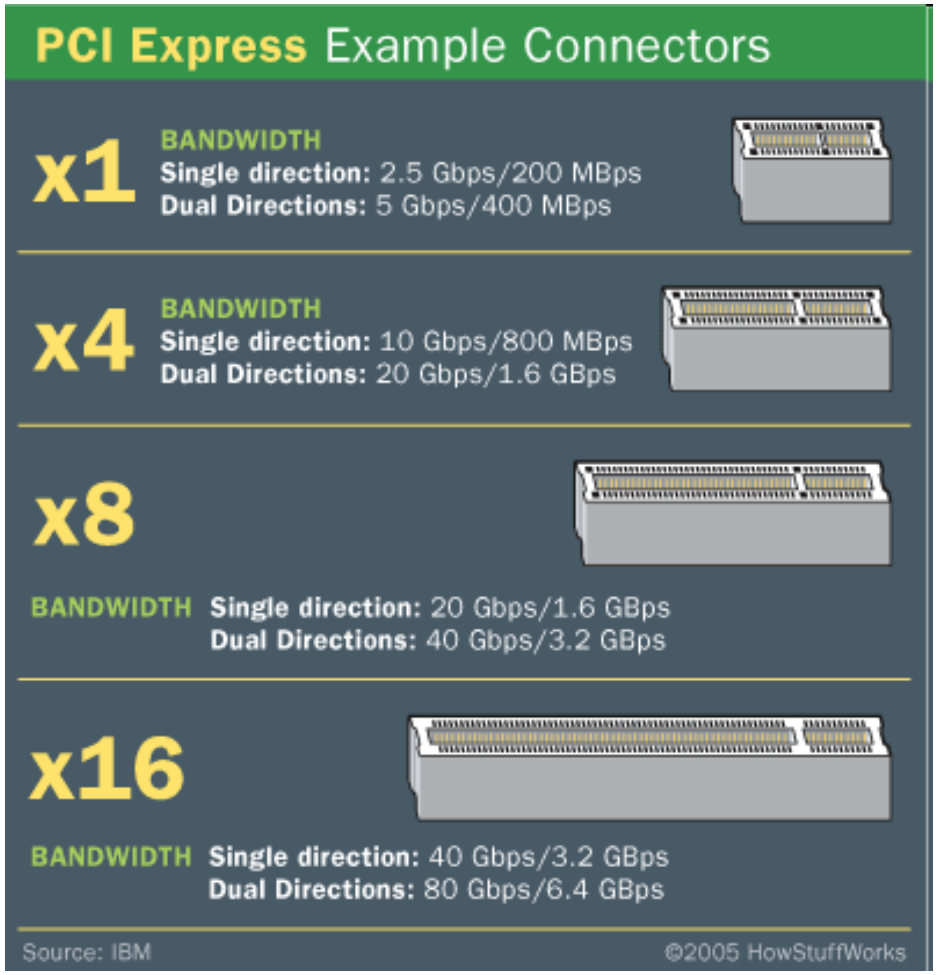
16 lane slot:

- **v1.x**: 4 GB/s
- **v2.0**: 8 GB/s
- **v3.0**: 16 GB/s

PCI-E links and lanes



PCI-E connectors



ARM Bus

- Introduced by ARM Ltd in 1996
- Widely used as the on-chip bus

ARM BUS

- AMBA = ARM Memory Bus Architecture
- AHB = ARM High performance Bus
- APB = ARM Peripheral Bus
- AMBA-AHB connects ARM core with memory, external DRAM
- AMBA-APB interfaces ARM core with external low-speed I/O devices using AMBA-APB bridge

AMBA

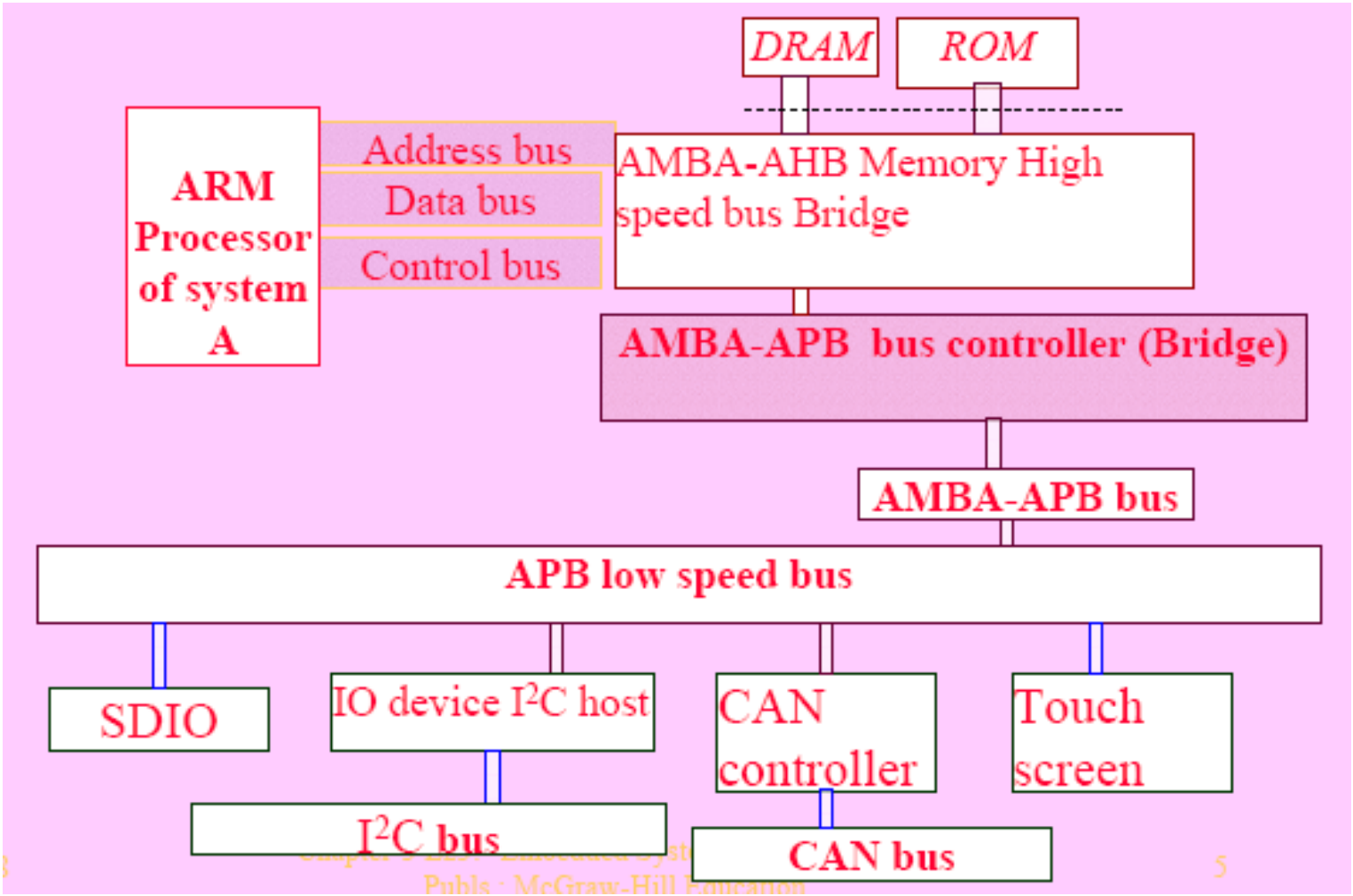
- AMBA-AHB connects to 32-bit data and 32-bit address at high speed
- AHB maximum bps bandwidth is 16 times ARM processor clock
- AMBA-APB bridge is used to communicate AHB bus to APB bus
- The bridge communicates to memory through AMBA-AHB

APB bus

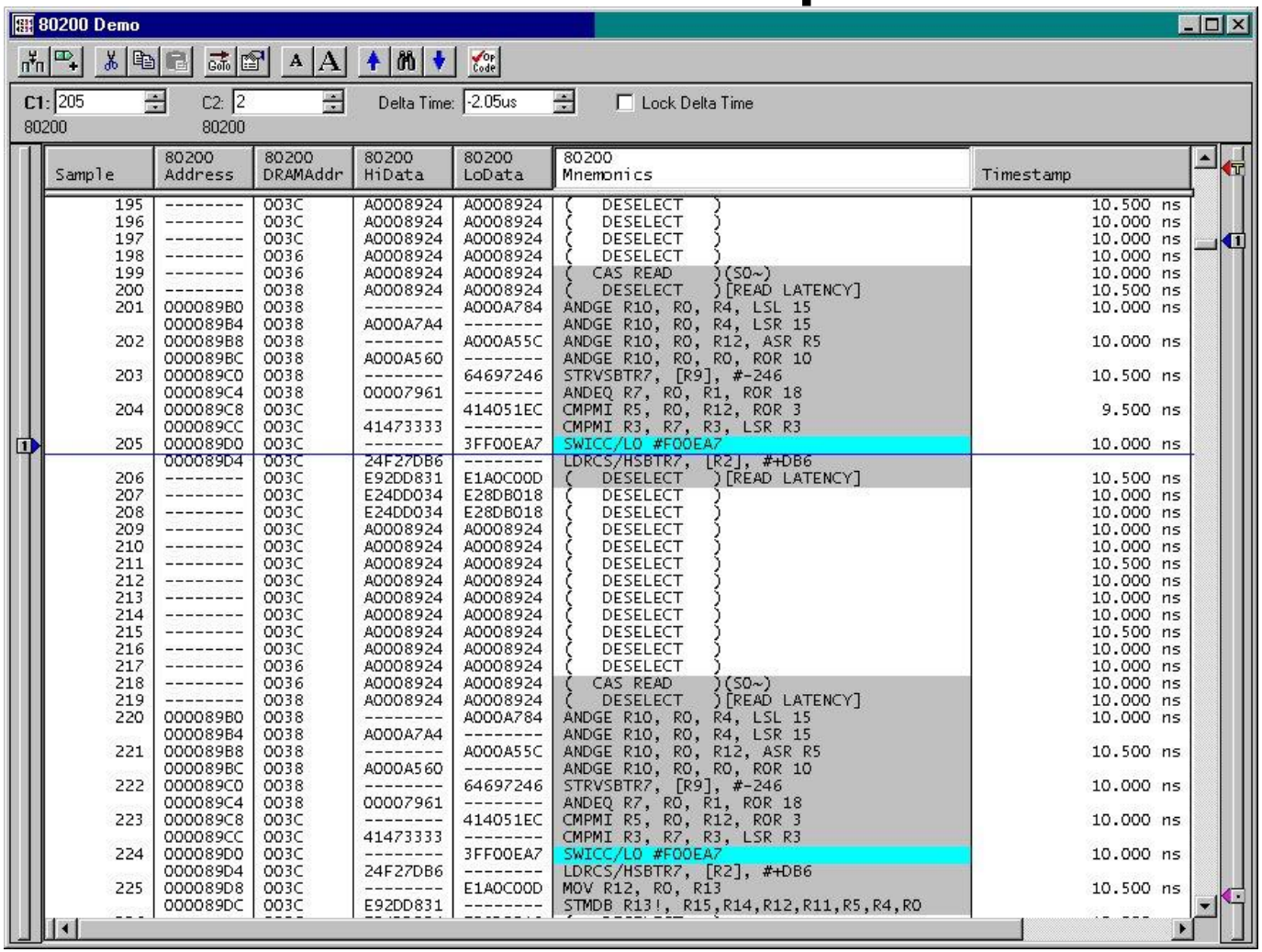
Connect:

- I^2C
- Touch screen
- SDIO
- MMC (multimedia-bus)
- USB
- CAN bus

ARM BUS


DRAM
ROM
ARM Processor of system A
Address bus
Data bus
Control bus
AMBA-AHB Memory High speed bus Bridge
AMBA-APB bus controller (Bridge)
AMBA-APB bus
APB low speed bus
SDIO
IO device I2C host
CAN controller
Touch screen
I2C bus
CAN bus

Logic analyzer to capture ARM instruction sequence



Sample	80200 Address	80200 DRAMAddr	80200 HiData	80200 LoData	80200 Mnemonics	Timestamp
195	--------	003C	A0008924	A0008924	(DESELECT)	10.500 ns
196	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
197	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
198	--------	0036	A0008924	A0008924	(DESELECT)	10.000 ns
199	--------	0036	A0008924	A0008924	(CAS READ)(S0~)	10.000 ns
200	--------	0038	A0008924	A0008924	(DESELECT)[READ LATENCY]	10.500 ns
201	000089B0	0038	--------	A000A784	ANDGE R10, R0, R4, LSL 15	10.000 ns
	000089B4	0038	A000A7A4	--------	ANDGE R10, R0, R4, LSR 15	
202	000089B8	0038	--------	A000A55C	ANDGE R10, R0, R12, ASR R5	10.000 ns
	000089BC	0038	A000A560	--------	ANDGE R10, R0, R0, ROR 10	
203	000089C0	0038	--------	64697246	STRVSBTR7, [R9], #-246	10.500 ns
	000089C4	0038	00007961	--------	ANDEQ R7, R0, R1, ROR 18	
204	000089C8	003C	--------	414051EC	CMPMI R5, R0, R12, ROR 3	9.500 ns
	000089CC	003C	41473333	--------	CMPMI R3, R7, R3, LSR R3	
205	000089D0	003C	--------	3FF00EA7	SWICC/LO #F00EA7	10.000 ns
	000089D4	003C	24F27DB6	--------	LDRCS/HSBTR7, [R2], #+DB6	
206	--------	003C	E92DD831	E1A0C00D	(DESELECT)[READ LATENCY]	10.500 ns
207	--------	003C	E24DD034	E28DB018	(DESELECT)	10.000 ns
208	--------	003C	E24DD034	E28DB018	(DESELECT)	10.000 ns
209	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
210	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
211	--------	003C	A0008924	A0008924	(DESELECT)	10.500 ns
212	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
213	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
214	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
215	--------	003C	A0008924	A0008924	(DESELECT)	10.500 ns
216	--------	003C	A0008924	A0008924	(DESELECT)	10.000 ns
217	--------	0036	A0008924	A0008924	(DESELECT)	10.000 ns
218	--------	0036	A0008924	A0008924	(CAS READ)(S0~)	10.000 ns
219	--------	0038	A0008924	A0008924	(DESELECT)[READ LATENCY]	10.000 ns
220	000089B0	0038	--------	A000A784	ANDGE R10, R0, R4, LSL 15	10.000 ns
	000089B4	0038	A000A7A4	--------	ANDGE R10, R0, R4, LSR 15	
221	000089B8	0038	--------	A000A55C	ANDGE R10, R0, R12, ASR R5	10.500 ns
	000089BC	0038	A000A560	--------	ANDGE R10, R0, R0, ROR 10	
222	000089C0	0038	--------	64697246	STRVSBTR7, [R9], #-246	10.000 ns
	000089C4	0038	00007961	--------	ANDEQ R7, R0, R1, ROR 18	
223	000089C8	003C	--------	414051EC	CMPMI R5, R0, R12, ROR 3	10.000 ns
	000089CC	003C	41473333	--------	CMPMI R3, R7, R3, LSR R3	
224	000089D0	003C	--------	3FF00EA7	SWICC/LO #F00EA7	10.000 ns
	000089D4	003C	24F27DB6	--------	LDRCS/HSBTR7, [R2], #+DB6	
225	000089D8	003C	--------	E1A0C00D	MOV R12, R0, R13	10.500 ns
	000089DC	003C	E92DD831	--------	STMDB R13!, R15,R14,R12,R11,R5,R4,R0	

Figure 2.16 A Tektronix logic analyzer with optional software setup to capture and disassemble an ARM processor's instruction execution sequence. This logic analyzer is actually an embedded system

Other Bus interfaces

- EISA bus
  - 32 bit, asymmetric I/O channel
  - up to 33 MB/s data transfer rate
  - 4GB address space, 8 DMA chanels
  - backward compatible with ISA
- Futurebus+
  - Designed by IEEE896 committee
  - 64 bit and 160 MB/s

Other Bus interfaces

- SCSI (small computer System Interface) bus
  ANSI standard
  4MB – 10 MB per second data transfer rate
  mainly for computer to other devices,
  WIDE SCSI uses wider cable

- Ultra SCSI
  Support from 20MB – 80 MB per second

- TURBO Channel bus
  32 bit, asymmetric synchronous I/O channel
  12.5 – 25 MHz data transfer rate
  Developed by DEC (digital)

Other Bus interfaces

- XMI Bus

  64 bits addressing

  Upto 100 MB per second
- IEEE-796 (Multi-bus)

  introduced by Intel for multiprocessors on the same board, support 16 bit data and 24 bit address buses
- VME bus (Euro standard)

  Similar to Intel Multi-bus, 24 bit address bus with 8/16/32 data buses

Questions?