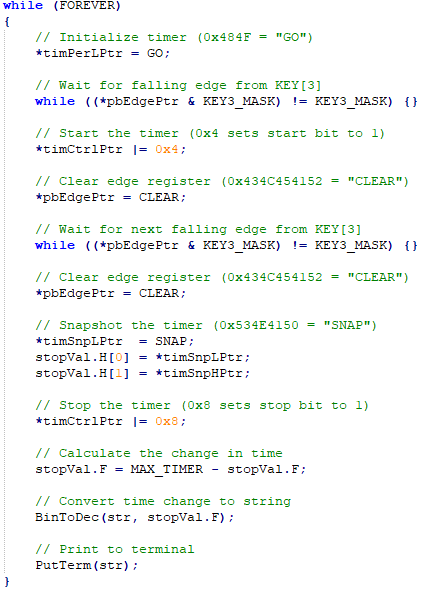
**Timers and Timing:**

Nios:

Down counter

Control.CONT bit:

1 = Counter is continuous

0 = Stop and hold new value

Snapshots:

When counter ends:

Issue IRQ, Reset processor (watchdog), Generate external pulse, do nothing.

System Clock:

Fixed period, Start/Stop control bits off, Control.CONT = 1, Control.START = 1.

Interval Timer:

Same as system clock with writable period enabled

Registers:

Offset by 4 bytes but are only 2-bytes

Treat as alt\_u16

C: separated by two halfwords (even)

EIA: 4 bytes

Use ldhuio to load timer reg into GPR

**Exception Handling:**

Registers:

Ctl0: EH = All exceptions, PIE = HW Interrupts

Exception: Copies ctl1 = clt0 to save state

ERET: Copies ctl0 = ctl1 to restore state

Ctl3: 1 = enabled, 0 = disabled

Ctl4: 1 = pending, 0 = no IRQ

Builtins:

\_\_builtin\_rdctl (int) : rdctl instr in ASM

Args = ctrl reg to read

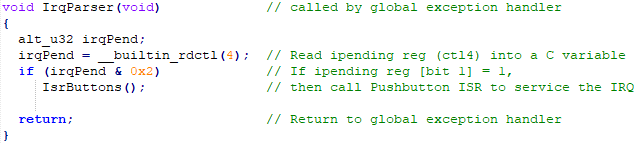
Retn = contents of ctrl reg

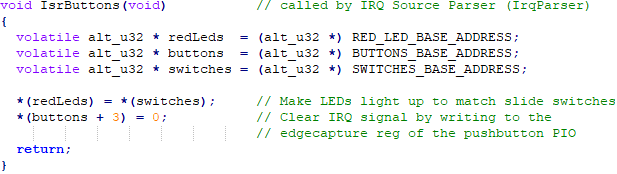
\_\_builtin\_wrctl (int, int) : wrctl instr in ASM

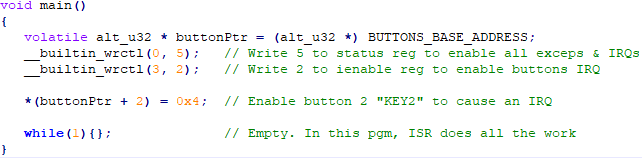
Args = ctrl reg to write, val to be written

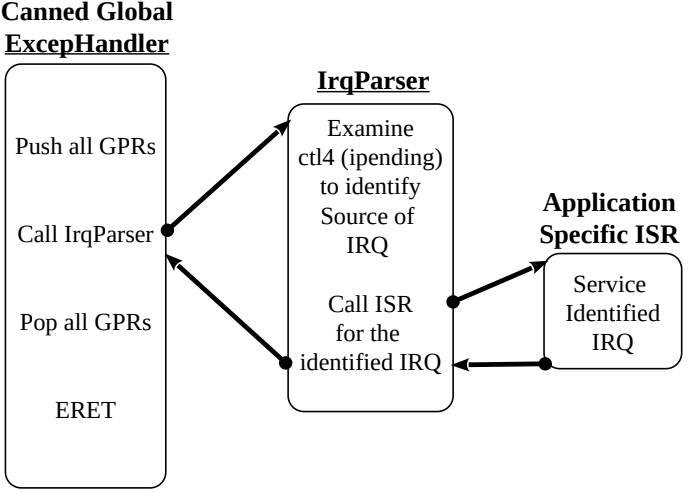
Nios uses a Non-vectored exception controller:

So a generic exception handler is used to call , an IRQ parser. Priority defined by SW.





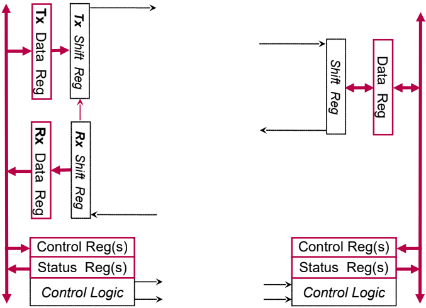




**Serial I/O Devices:**

**SPI: Serial Peripheral Interface**

Properties: Full Duplex, Synchronus, Master- , Slave Protocol **(No Overhead)**



SCLK

SS

MOSI

MISO

Signals:

MISO: Master-In-Slave-Out

MOSI: Master-Out-Slave-In

SCLK: Serial Clock: Shift output on one edge , and latch input on other edge

SCLK = Avalon Clock / 2^n

AC = 50 Mhz : SCLK = 50, 25, 12.5, etc.

Must be slower than Slave

SS: Slave select

Polarity: Quiescent state of SCLK

0: Quiescent = Low = 0; Lead edge = Rise edge

1: “ “ = High; Lead edge = Fall edge

Phase: What happens on each SCLK edge

0: Latch input on lead, change output on trail

1: Change output on lead, latch input on trail

Errors:

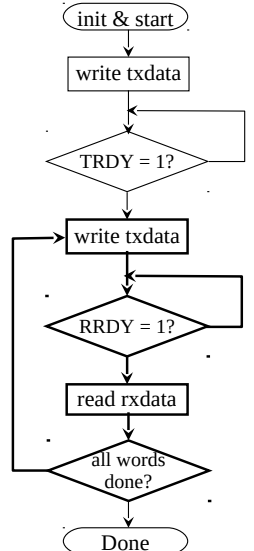
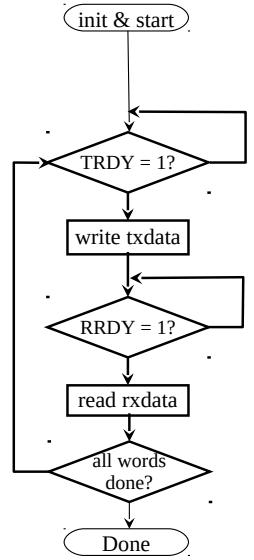
TOE: Transmit Overrun Error – CPU too fast

TRDY must be set to 1 b4 transmitting

ROE: Receive Overrun Error – CPU too slow

RRDY must be set to 1 b4 receiving

**Simple Full Duplex Better Full Duplex**



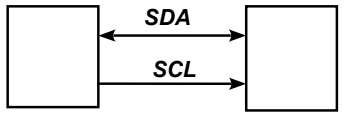
**I2C: Inter-Integrated Circuit Bus**

Properties: Half-Duplex, Synchronus, Master-Slave Protocol (technically multi master)

Signals:

SDA = Serial Data Line (bidirectional)

SCL = Serial Clock Line (unidirectional)



S

M

Multi Master Bus: Electrically handles simultaneous masters through Wired-AND Bus

UART Pins:

RXD = Receive Data

TXD = Transmit Data

Optional Flow Control Pins:

RTS = Request to send

CTS = Clear to send

Wired-AND: Quiescent state = 1 (pull-up res)

0 is dominant: single zero overrides all 1’s

1 is recessive : only exists if no zeros

Bit Signaling Convention:

1 data bit lasts 1 SCL clock period

Transm. change SDA value only when SCL = 0

Recev. read SDA on rising edge of SCL

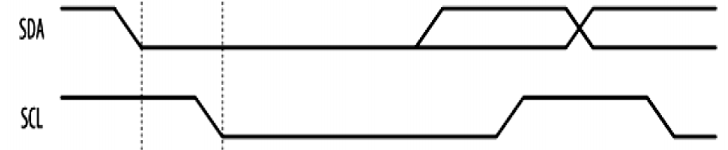
Starting: Quiescent State: SDA & SCL = 1

Start Bit: (S-Bit)

Master asserts SDA = 0 **while SCL is 1**

Master asserts SCL = 0

Master asserts 1st SDA b4 next SCL rise edge

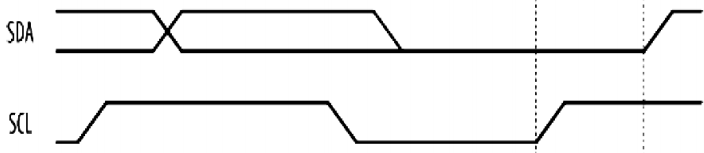


Terminating: Stop Bit: (P-Bit)

Master asserts SDA = 0 while SCL = 0 (norm)

Master asserts SCL = 1

Master Asserts SDA = 1 **while SCL is 1**



**I2C Continued: Byte Signaling Convention:**

All transfers in multiples of a byte (msb 1st)

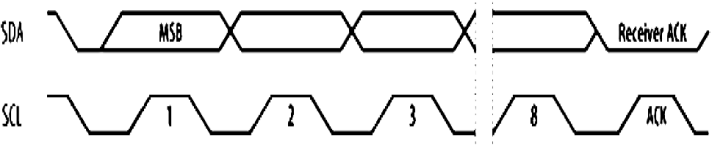
9th bit is ACK from recv.

Trans. release SDA bus (floats to 1)

Recv. asserts SDA = 0 to ack

Master asserts SCL = 1 and reads ack

If Recv. sends SDA = 1, then NACK



Contents:

S-bit: Master 🡪 Slaves

Addr/Dir Byte: Master 🡪 Slaves

Data Bytes: Trans. 🡪 Recv.

P-bit: Master 🡪 Slaves

R/W bit: 1 = Read, 0 = Write

New Master Bus Arbitration:

Carrier Sense: Listen for quiescent bus

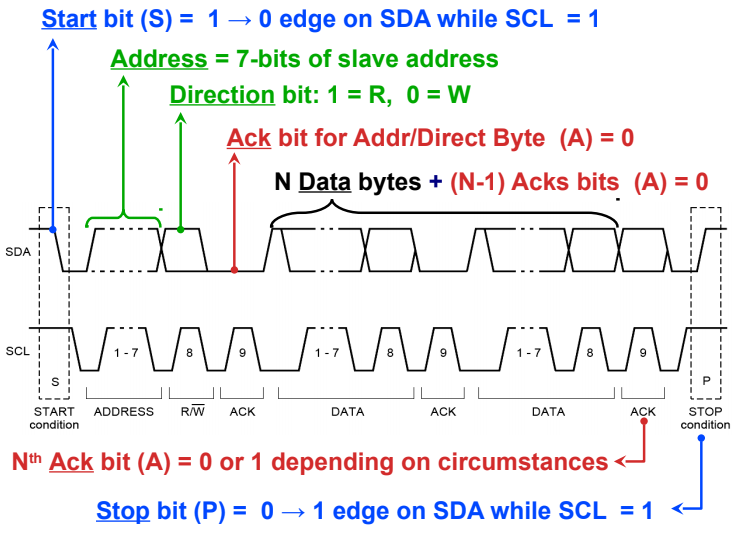
(SDA = 1 & SCL = 1)

Collision Avoidance: Listen while transmitting

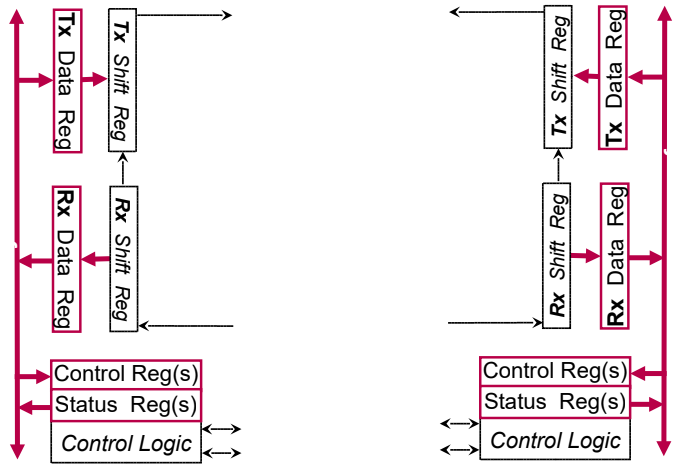
If SDA = 0 while sending 1, Stop and try later

0 will keep going

Summary:



**UART: Universal Asynchronous Receiver/Transmitter**



RTS

CTS

RxD

RxD

TxD

TxD

Protocol: Trans. & Recv. must agree on…

Bit-Clock freq. = Baud Rate

# of bits per word (short to prevent drift)

Endian order (usually lsb 1st)

Parity – even, odd, or none (E, O, N)

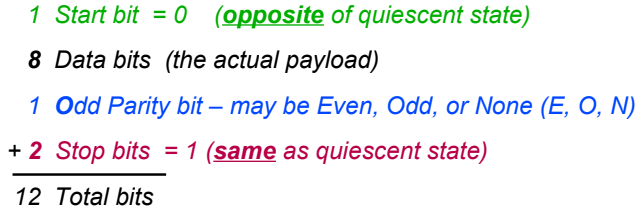
Idle or “quiescent” state 🡺 1

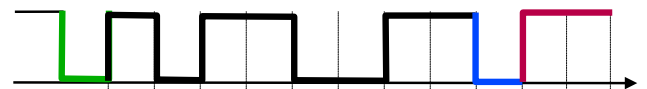
Framing bits to identify a word

1 # of start-bits : opposite of Q-state = 0

1 or 2 # of stop-bits : same as Q-state = 1

Data Formatting:





Payload efficiency = 8 / 12 = 66.7%

Message Synchronization:

Recv. over-samples, 8x to 16x the bit-clock

Timing Error:

Occurs when CLK freq. are too far out of sync

Need to keep UART words short (~ 1Byte)

Framing Error:

Occurs when Recv. gets confused about start , of message

If the expected stop bit is a zero then

Recognizes the error (no garuntee)

Break Error:

Occurs if Recv. sees too many 0’s in a row.

May be accompanied by a framing error

May indicate transmitter stuck at zero

Or transmitter can intentionally send a

“Break Character”

Break Character = long string of zeros:

Sent to stall mid data stream or

Flag an unrecoverable error and abort

Overrun Errors: ROE and TOE

Same as SPI

Parity Error: Recv. par bit != expected par bit

Noise Error: channel is too noisy.

Receiver bit samples do not agree

Nios UART Params:

Word is called a “Character”

Set at configuration

7 bits = ASCII character

8 bits = Byte or ASCII w/ parity

9 bits = Byte w/ parity

Transfer bit order = LSB first

**JTAG: Joint European Test Action Group**

Why? Testing dense circuit boards

Couldn’t test using nails as circuits got smaller

Boundary Scan Architecture:

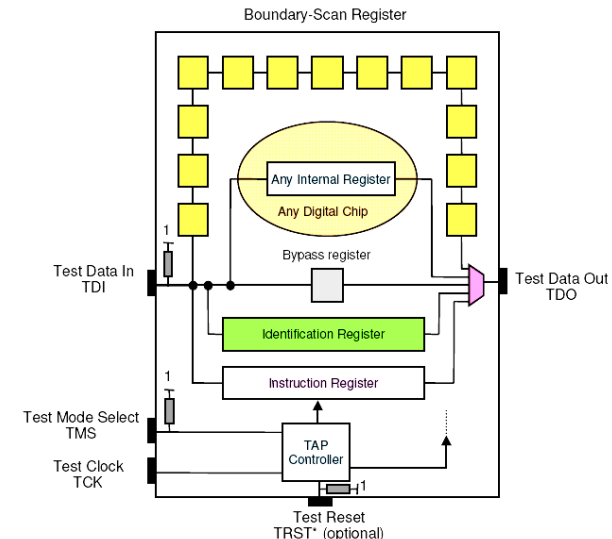
Boundary Scan Cell at each pin

Boundary Scan Register (BSR) = set of all cells

Test Access Port (TAP) compromising

Four required TAP I/O pins

TAP controller state machine



TAP Signals:

TCK = Test Clock: Triggers shift reg and tap ctrlr

TDI = Test Data In = Values to be shifted in chip

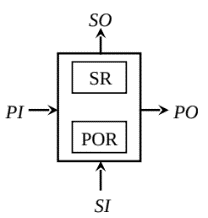
TDO = Test Data Out = “ ” to be shifted out chip

TMS = Test Mode Select: Next-State cmd to TCtl

TRST = Test Reset (Optional)

Reset command for TCtl State Machine

Can also be done by sequence of TMS , transitions

JTAG Chain: multiple chips daisy chained

Connect TDO of one chip to TDI of next chip

Broadcast TCK, TMS, and TRST (if needed)

TAP Registers:

Instruction Register: IR

n-bit (n > 2) current inst. For TAP ctrlr

Really two registers:

Scan: receives new inst. From TDI

Hold: Holds curr. Inst.& provides output , signals to selected data registers

Data Registers:

BSR: Already defined

Bypass Register:

1-bit reg conn. From TDI to TDO

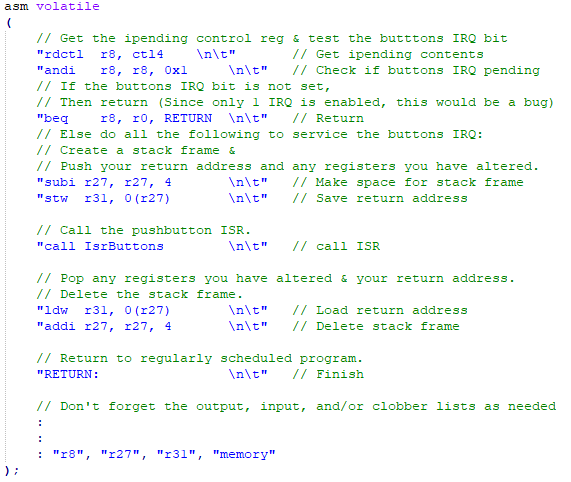
Passes each bit shifted in w/o action

Easily test some chips

BSR Functional Modes:

Normal: Pas data PI to PO

Capture: Latch PI to Shift Reg (SR)

 Serial Shift:

Pass SR out SO, to next cell in chain

Latch SI into SR from previous cell in chain

Update: Drive parallel output reg POR out PO

**Generic Parallel Busses:**

Intra-System: Within one computer

Fast, small runs, shared parallel DC bus

Addr, Data & Ctrl lines

Inter-System: Btw Independent computers

Slower, Long runs, tends to be serial

**Generic Bus Arbitration:**

Participants: Bus Control Unit (BCU) & Bus Units

3 basic control signals:

REQ = Bus Request (BU to BCU)

GNT = Bus Grant (BCU to BU)

BSY = Bus Busy (Current Master to BCU)

Independent Request:

N REQ and GNT lines and 1 BSY line (wired or)

ADV: Immediate, proggr. priority (fair), Only 3 pins on BU

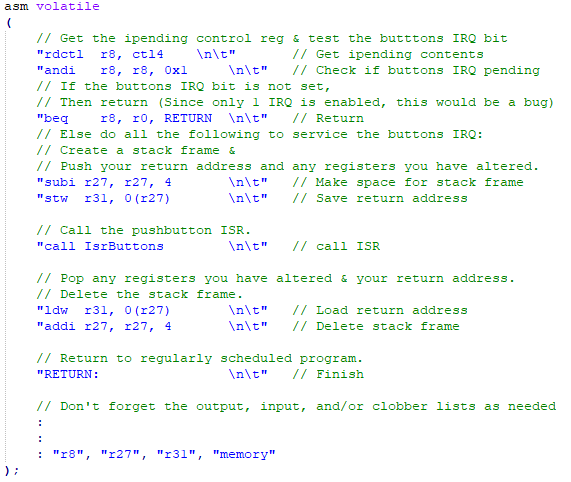
DISV: O(N) Pins on BCU and N Lines 4 R&G

N must be small

Poll Counter:

N Poll lines (GNT) and 1 BSY & REQ lines

ADV: O(logN) bus lines and pins on BCU

 Can be fair

DISV: N bounded by number of poll lines

O(logN) pins per BU

Delay when cycling addrs

Daisy Chain:

1 BSY, REQ, & GNT lines

ADV: 3 ctrl lines and BCU pins and 4 BU pins

DISV: 1 faulty BU can break others down line

Delay to ripple GNT down line

Not fair

**DMA: Direct Memory Access**

Responsibilities:

CPU: Pass block params to DMAC, Kickstart DMA transfer, & arbitrate bus cycles

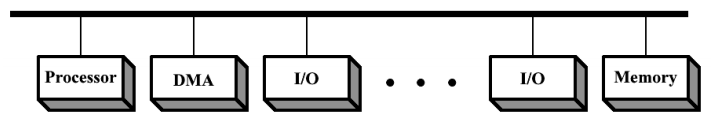
DMAC: Transfer data, Update addr and counter, & inform CPU when finished, (IRQ)

Detached:

DMAC acts as a surrogate CPU for trans.

I/O dev 🡨> DMAC 🡨> Memory

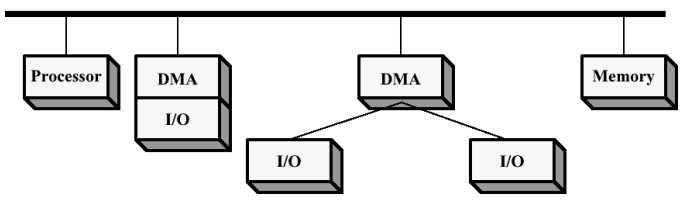
2 ops per word (inefficient, but cheap)



Integrated:

Devices attached directly to DMAC

1 op per transfer

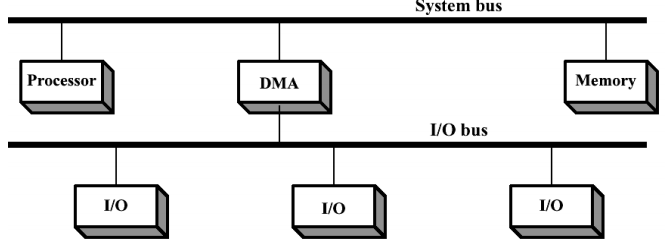


DMA I/O Bus:

Expandable ver. Of Integrated DMA

1 bus op per transfer

Best for slower devices



**Bus Cycle Arbitration:**

1 Cycle Stealing:

Cycle 1: DMAC issues DMA\_REQ & CPU asserts DMA\_ACK for next cycle

Cycle 2: DMAC uses bus for one cycle

Perf: DMAC uses bus every 2nd cycle

Causes occasional slow down of CPU

2 Transparent DMA:

Cycle 1: DMAC issues DMA\_REQ

CPU decides if DMAC gets Bus 4 next cycle

Cycle ?: DMAC uses bus for 1 cycle

Perf: DMAC uses bus no more than every 2nd cycle

CPU never has to stall

3 Block Mode DMA:

Cycle 1: DMAC issues DMA\_REQ & CPU asserts DMA\_ACK for next cycle and holds it high

Cycle 2: DMAC uses the bus for an entire block’s worth of cycles

Cycle ?: DMAC clears DMA\_REQ, CPU clears DMA\_ACK

Perf: DMA use every cycle

CPU stalls for whole block

CPU performance still better than prog I/O

Impact of Pipelining:

Cycle: DMAC and CPU op at half speed

Trans: CPU keeps bus forever, DMAC gets nothing

Burst: CPU stalls completely

Impact of Cache:

Prog I/O: At least 2 bus cycles to ld and st data

I/O rate is several instruct cycle / word

DMA I/O: each word = 1 bus cycle

I/O rate is several word / instruct cycle

Block I/O: DMA = higher I/O throughput than PIO

DMA = higher CPU utilization than PIO

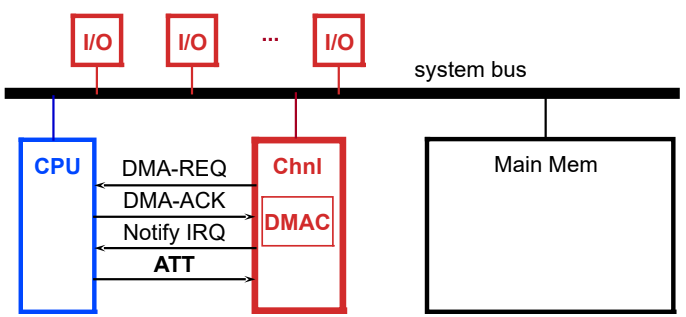
**IOP: I/O Processor** Programmable DMAC

Communications:

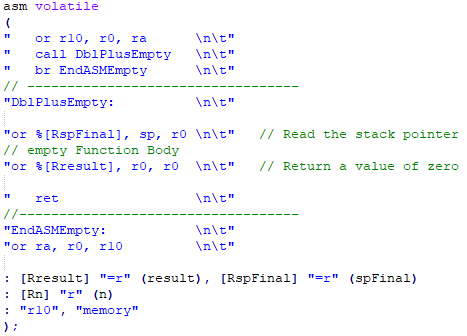
CPU initiates I/O operation storing the starting addr of the channel program, storing parameters the channel will need, and issuing an attention signal ATT to wake up Channel.

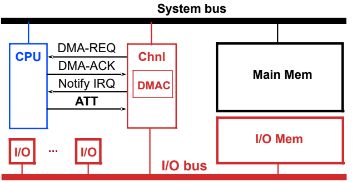
IOP then fetches I/O program addr from mailbox, load into its own PC and executes the I/O program by setting up parameters, transferring control to DMAC and resume ops when DMAC is done and store remaining status info into mailbox, then issues the notify IRQ to CPU

Single Bus:

Significant competition for memory bus, Greatly mitigated by caches

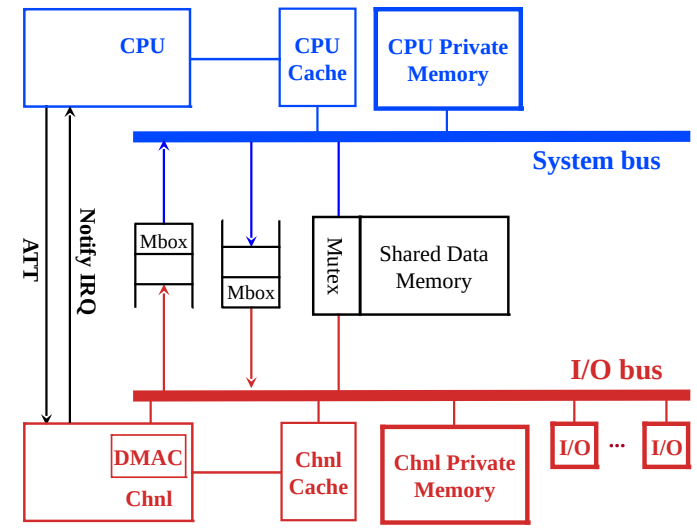
Dual Bus:

 Channel has its own program memory, reduces competition for data memory space, Greatly mitigated by caches



Shared Memory:

Command & control passed through dedicated mailboxes, shared data memory for i/o data



**PCI: Peripheral Component Interconnect**

Features:

Synchronus single phase clock

Bus width 32 or 64 bits

Multiplexed Address/Data lines (AD)

Terms:

Initiator = Bus Master

Target = Active Bus Slave

Transaction = 1 transfer of data (may contain multiple words)

(supports burst ops)

# = Active low signal (IRDY# or TRDY#)

CK = PCI Bus Clock

Assert & clear all signals on falling edge

Sense all signals on rising edge

C/BE# = Command/Byte Enable (4-bits or 8-bits)

Addr Phase: Sends cmd to target

Data Phase: Indicates which bytes of AD lines are valid

Frame#: Asserted by initiator at start and cleared at start of

final data phase

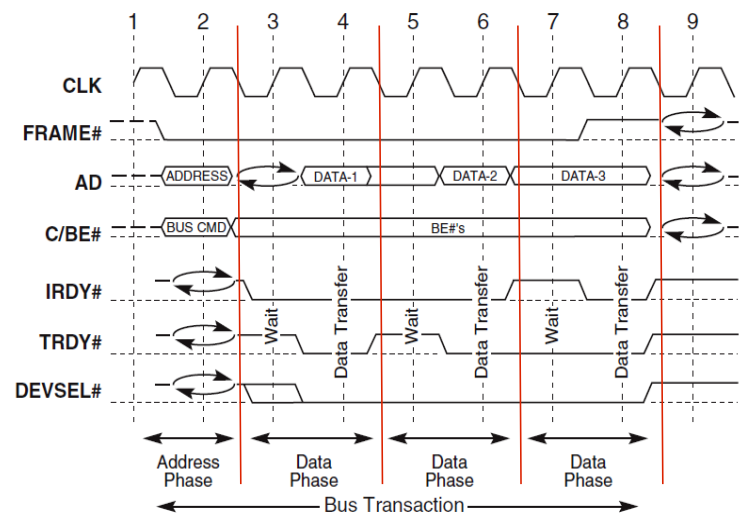
DEVSEL#: Asserted by target when recognizes its addr

Indicates target dev has been selected

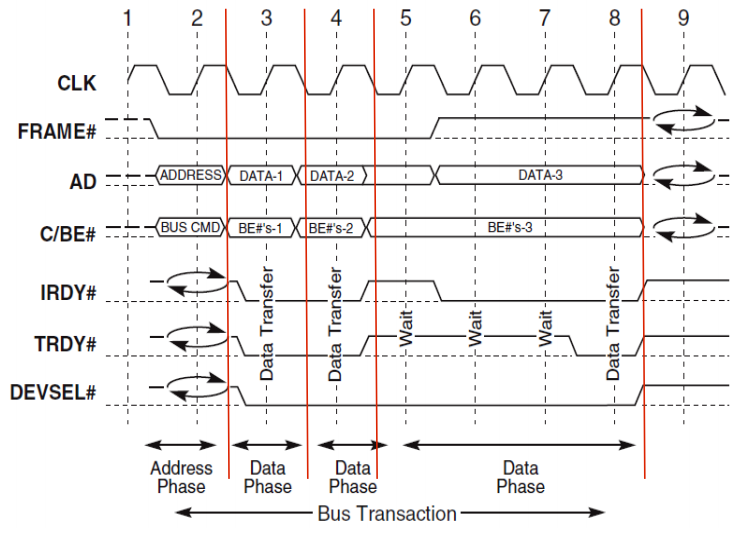
IRDY#: Initiator Ready: Read = ready to recv., Write = AD valid

TRDY#: Target Ready: Read = AD valid, Write = ready to recv.

Read example:



Write example:



Arbitration:

Independent request scheme (REQ# and GNT#)

No BSY line, uses {FRAME# and IRDY#} (either low bus = busy)

Uses Hidden Arbitration

Arbitrates next master during current transaction

