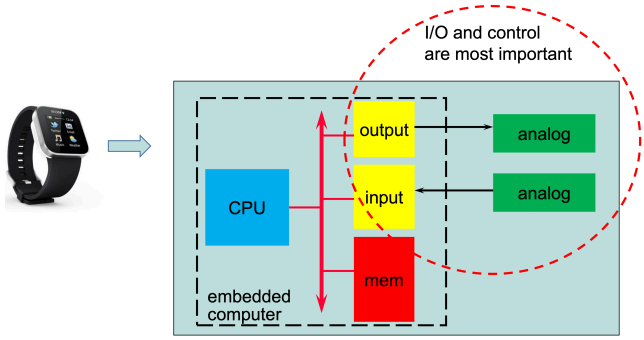
Catalogue

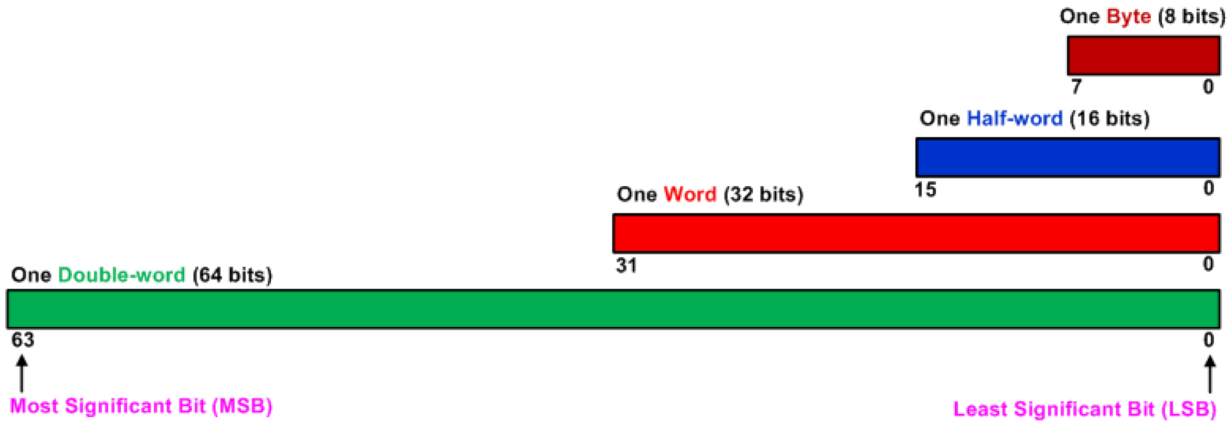
Lecture 1: Intruduction

Embedded System

- An embedded system is an application that contains at least one programmable computer.
- Embedded systems are information processing systems that are embedded into an enclosing product.
- An embedded system may or may not have OS.

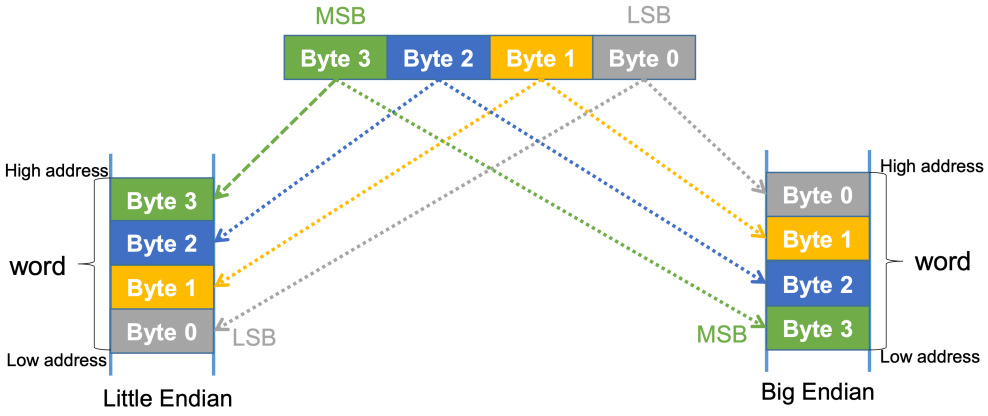


Value Notations (32 bit processor)



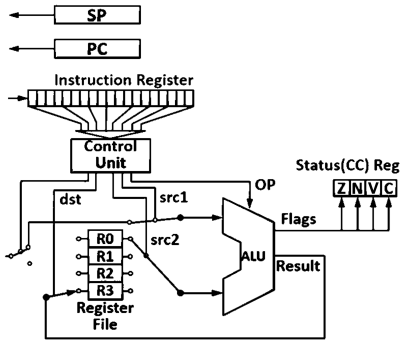
Little Endian vs Big Endian

- Little endian: LSB of a word is at the least memory address.
- Big endian: MSB of a word is at the least memory address.



Lecture 2: STM32 MCU & GPIO

CPU

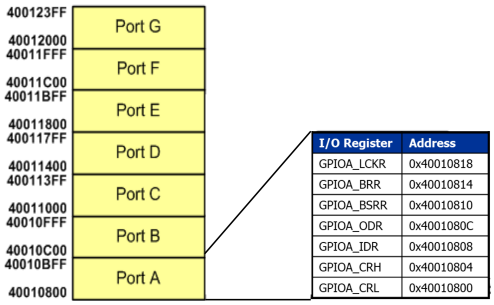


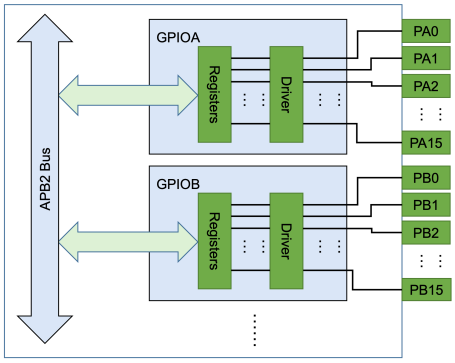
- ALU
  - operands (stored in registers and data memory)
  - operation
  - flag (stored in Program Status Register): **Z**ero, **N**egative, o**V**erflow, **C**arry
  - result
- Register File
  - low register (r0 - r7): can be accessed by any instruction
  - high register (r8 - r12): can only be accessed by some instructions
  - stack pointer (r13)
  - link register (r14): stores the return address for function calls
  - program counter (r15): memory address of to be executed instruction

    - program status register
- Control Unit

GPIO

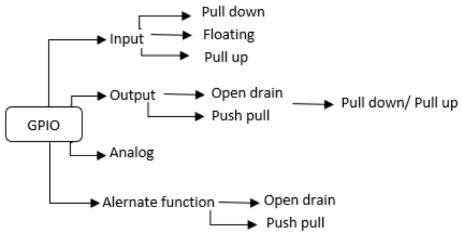
- Processor can directly access processor registers; accesses peripheral registers via memory mapped I/O.
- Each register has a specific memory address, Register Mapping assigned a name to each register address.





- GPIO Mode

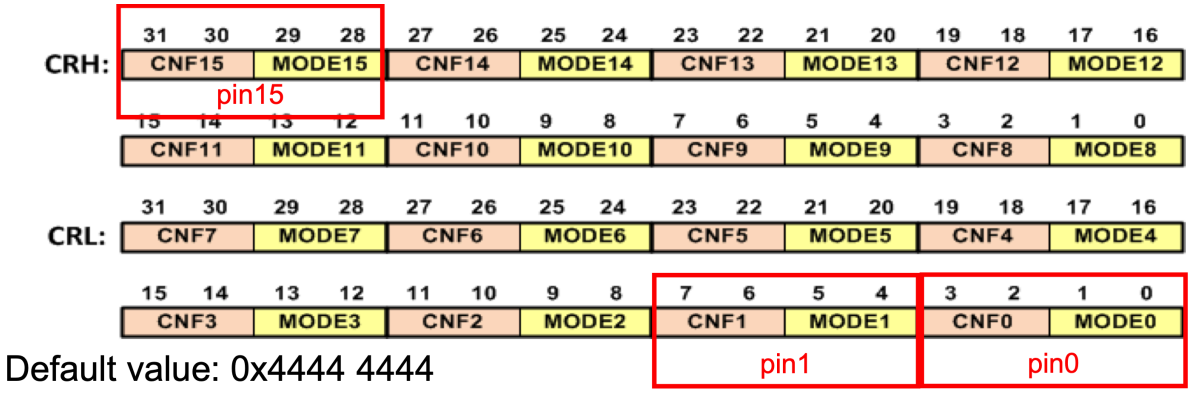
GPIO Mode	Usage
Floating input (reset state)	Completely floating, and the state is undefined
Input with pull-up	With internal pull-up, defaults to high level (button)
Input with pull-down	With internal pull-down, defaults to low level
Analog mode	ADC, DAC
General purpose output Open-drain	Software I2C, SDA, SCL, etc
General purpose output push-pull	Strong driving capability, general-purpose output (LED)
Alternate function output Open-drain	On-chip peripheral functions (hardware I2C, SDA, SCL pins, etc)
Alternate function output Push-pull	On-chip peripheral functions (SPI, SCK, MISO, MOSI pins, etc)



    - default: floating input
    - input: pull-up/pull-down
    - output: push-pull
- Programming GPIO
    - configure the GPIO mode: set CRL/CRH to configure input/output mode

- set the output status: set ODR to configure output status
- read the input status: set ODR to configure input with pull-up/pull-down; read from IDR to get input status

- GPIO Registers (each group GPIO ports has 7 registers)
  - CRL/CRH (32-bit configuration register): every 4 bits combine to represent a pin



    common settings for each pin (default = 0x4):

    - 0x3 = 0011: CNF = 00 (pushpull), MODE = 11 (output)
    - 0x4 = 0100: CNF = 01 (Hiz), MODE = 00 (input)
    - 0x8 = 1000: CNF = 10 (pull-up/pull-down), MODE = 00 (input)

    example:

```
GPIOC -> CRH &= 0xFFF00FFF; // clear settings of PC11 and PC12
GPIOC -> CRH |= 0x00038000; // set PC11, PC12 as input and output, respectively
GPIOC -> ODR |= 1 << 11; // set PC11 as input with pull-up
```

  - IDR/ODR (32-bit input/output data register): the high 16 bits are reserved, each one of the low 16 bits represents a pin

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy:** Port input data (y= 0 .. 15)
These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

for LED, set ODR bit to **low** to **turn on** the LED.

for KEY0/KEY1, set **pull-up** input mode.

for KEY-WK, set **pull-down** input mode.

- BSRR (32-bit set/reset register)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

- BRR (32-bit reset register)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved

Bits 15:0 **BRy:** Port x Reset bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

- LCKR (32-bit locking register)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															LCKK
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved

Bit 16 **LCKK[16]:** Lock key
This bit can be read anytime. It can only be modified using the Lock Key Writing Sequence.
0: Port configuration lock key not active
1: Port configuration lock key active. GPIOx_LCKR register is locked until an MCU reset occurs.

LOCK key writing sequence:
Write 1
Write 0
Write 1
Read 0
Read 1 (this read is optional but confirms that the lock is active)

Note: During the LOCK Key Writing sequence, the value of LCK[15:0] must not change.
Any error in the lock sequence will abort the lock.

Bits 15:0 **LCKy:** Port x Lock bit y (y= 0 .. 15)
These bits are read write but can only be written when the LCKK bit is 0.
0: Port configuration not locked
1: Port configuration locked.

Lecture 3: ARM Assembly

Flags

- Numbers are stored in 2's complement representation.
- CPU doesn't know whether the number is signed or unsigned.
- Condition flags in PSR:
  - **N**egative: N = 1 if most significant bit of result (do not consider the potential carry bit) is 1
  - **Z**ero: Z = 1 if all bits of result (do not consider the potential carry bit) are 0
  - **C**arry (treat as unsigned numbers):
    - for unsigned additoin, C = 1 if carry takes place
    - for unsigned subtraction, C = 0 if borrow takes place
    - for shift/rotation, C = last bit shifted out
  - o**V**erflow (treat as signed numbers):
    - V = 1 if adding two same-signed numbers produces a result with the opposite sign
    - non-arithmetic operations doesn't touch V bit
- Most instructions update NZCV flags only if 'S' suffix is present (e.g. ADD - unchanged, ADDS - changed), while some instructions always update them (e.g. CMP).

Logic Instructions

- `AND r0, r1, r2`: bitwise and, `r0 = r1 and r2`
- `ORR r0, r1, r2`: bitwise or, `r0 = r1 or r2`
- `EOR r0, r1, r2`: bitwise exclusive or, `r0 = r1 xor r2`
- `BIC r0, r1, r2`: bit clear, `r0 = r1 and ~r2`
- `LSL r1, r2, num`: logical shift left, `r1 = r2 << num`
- `LSR r1, r2, num`: logical shift right, `r1 = r2 >> num`
- `ROR r1, r1, num`: rotate right

Addressing Mode

- Immediate addressing
- Register addressing
- Register indirect addressing (indexed): base + offset
  - `LDR r1, [r2, num]` (pre-index): load r2[num] to r1
  - `LDR r1, [r2, num]!` (pre-index): r2 += num, and then load r2[0] to r1
  - `LDR r1, [r2], num` (post-index): load r2[num] to r1, and then r2 += num

---

Lecture 4: Embedded C

Operation

- Bit manipulation
  - &: clear a bit
  - |: set a bit
  - ^: toggle a bit
  - shift: mul/div
- if-else/switch-case

  switch-case runs faster, because it will generate a forwarding table, in order not to go through all the branches.
- macro
  - preprocessor is executed before the compilation, for file inclusion, macro substitution, conditional compilation, and so on
  - macro is a fragment of code as a replacement in the source code

Data Type

- pointer
- array
- string
- struct
- union: members occupy the same memory space

Storage

- static variable
  - declared within body of a function: maintain the value between function calls, i.e., when calling a function secondly, the value is the same as which at the end of the first-called function
  - declared within a module but outside the body of any function: accessible by all functions within that module; independent with variables in other modules, even if they have the same name
- volatile variable

  the value may change outside the normal program flow, possibly, via an interrupt service routine or a hardware action
- malloc/free

  costly in time and space

Lecture 5: Interrupt

Subroutine

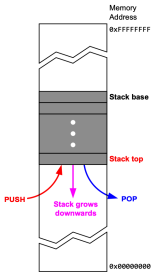
- call a subroutine: BL (branch and link)

```
LR = PC + 4 // save the return address in the link register (LR)
PC = label
```

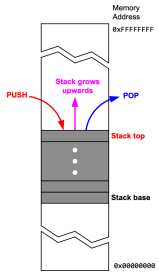
- back to caller: BX (branch with exchange)

```
PC = LR
```

- stack
  - descending/ascending

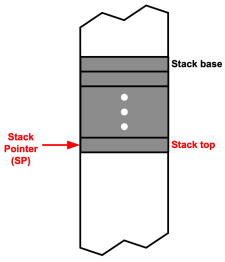


Descending stack: Stack grows towards low memory address

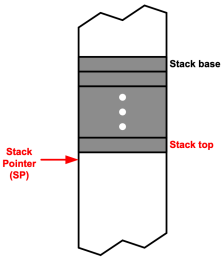


Ascending stack: Stack grows towards high memory address

- full/empty



Full stack: SP points to the last item pushed onto the stack

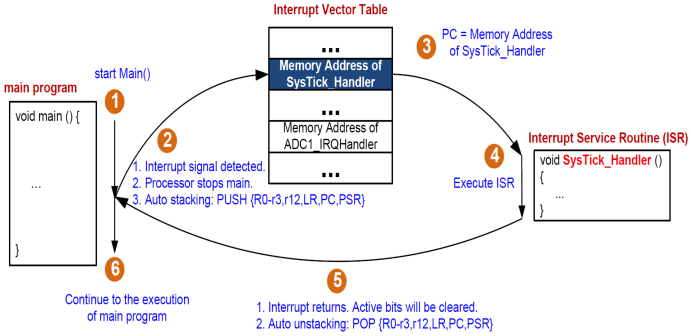


Empty stack: SP points to the next free space on the stack

- Cortex-M uses full descending stack

Interrupt

- Process
  - finish processing current instruction
  - save return address and register context to stack
  - run interrupt service routine
  - restore return address and register context from stack
  - resume main program
- Each interrupt has an ISR (Interrupt Service Routine)
- As for Cortex-M, an interrupt vector table in memory contains addresses of ISR
  - table elements located at fixed address
  - table elements store function pointers (address of ISR)



- Interrupts are managed by Nested Vectored Interrupt Controller (NVIC)
  - AIRCR register: set priority group

    - IPRx register: set priority value
        - smaller value means higher priority
        - fixed priority for Reset, HardFault and NMI; adjustable for the other interrupts
    - ISER/ICER registers: enable/disable interrupts
- Priority
    - preempt priority (first)
    - sub priority (second)
    - natural priority (third)
- Principle
    - higher preempt priority can interrupt ongoing lower preempt priority
    - same preempt priority
        - higher sub priority can not interrupt ongoing lower sub priority
        - when two interrupts occur at the same time, the one with higher sub priority is executed first
        - same sub priority: depend on which interrupt occurs first
- BL vs Interrupt

BL	Interrupt
Jumps to any location	Jumps to a fixed location
BL is used by the programmer in the sequence of instruction	hardware interrupt can come in at any time
cannot be masked	Can be masked (disabled)
Saves only LR register (Value of PC)	Saves CPSR, PC, LR, R12, R3, R2, R1, and R0.
CPU mode remains unchanged	CPU goes to Handler mode
On return, restores LR register (Value of PC)	On return, restores CPSR, R15, R14, R12, R3–R0.

Lecture 6: Serial Communication - UART

Communication Interfaces

- UART/USART
- SSI/SPI
- I2C
- USB
- Ethernet
- CAN

Serial/Parallel

Characteristics	Transmission Rate	Anti-Interference Ability	Communication Distance	I/O Resource Usage	Cost
Serial Transfer	Low	High	High	Low	Low
Parallel Transfer	High	Low	Low	High	High

Synchronous/Asynchronous

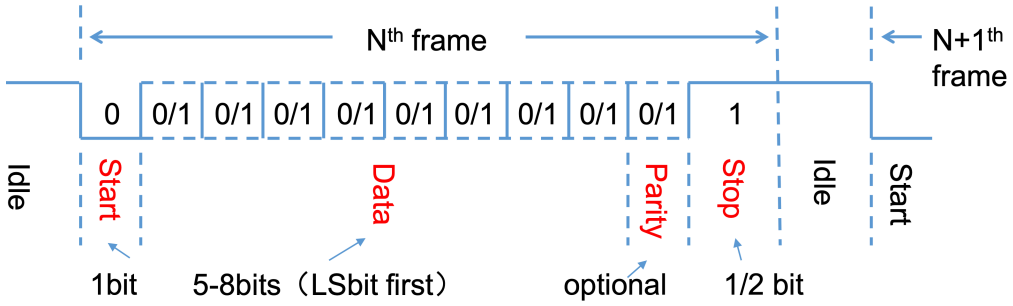
- Synchronous Communication (e.g. I2C, SPI, USART)
  - shares the same clock signal which is sent along with data
  - the device that generates the clock is called the master and other devices are slaves
- Asynchronous Communication (e.g. UART, USART)
  - sender provides no clock signal to receiver
  - relies on synchronous signals like start and stop bits within the data signal

Direction

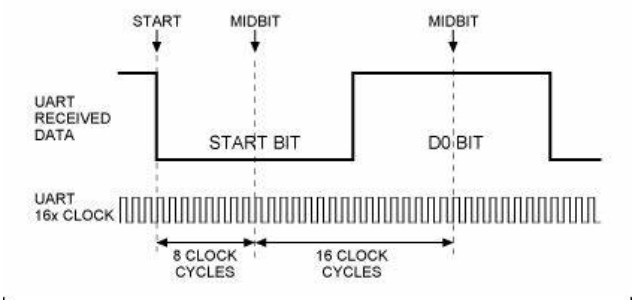
- simplex: data can only be transmitted in one direction always
- half-duplex: data can be transmitted in both directions but need time division (e.g. I2C)
- full-duplex: data can be simultaneously transmitted in both directions (e.g. UART, SPI)

UART

- frame format



  - ST: start bit = 0
  - SP: stop bit =1 to make sure that the next start bit makes a falling edge
- transmission speed of actual data = baud rate / number of bits in each frame
- synchronization of two ends: each clock of the end should be accurate to within about ±2.5%
- bit level reorganization: oversampling with receiver sample clock being 16x faster than baud rate



  - at START, discover 1 -> 0
  - after 8 clock cycles, reach the first MIDBIT, do sampling (and find that the first 8 bits are 0, which should be a start signal)
  - after 16 clock cycles, reach the second MIDBIT, do sampling (the first data bit of frame)
- error detection: add a parity bit
- RS232 defines the electrical and mechanical characteristics
- STM32 USART

  Baud rate register (USART_BRR): the baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the Mantissa and Fraction values of USARTDIV
  $baud = \frac{f_{USART\ input\ clk}}{16 \times USARTDIV}, USARTDIV = DIV\_Mantissa + \frac{DIV\_Fraction}{16}$

Lecture 7 & 8: Timer

Clock Tree

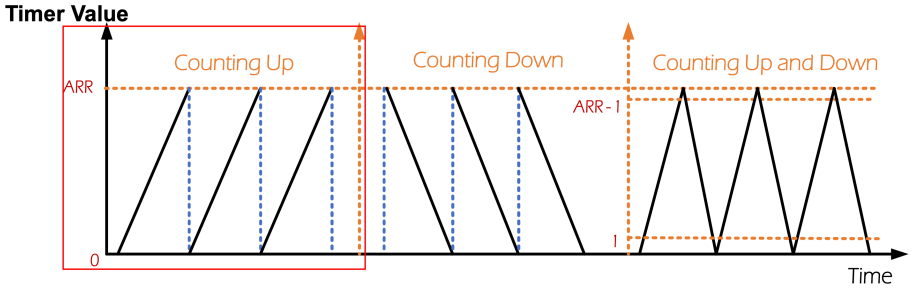
- clock source
  - Quartz Crystal Oscillators
    - high accuracy
    - stable frequency
  - RC/LC/RLC oscillators
    - simplicity
    - low cost
  - PLL
- STM32F103 clock config

Source	Freq.	Material	Usage
HSE	4~16MHz	Oscillator	SYSCLK
LSE	32.768KHz	Oscillator	RTC
HSI	8MHz	RC	SYSCLK
LSI	40KHz	RC	RTC/IWDG

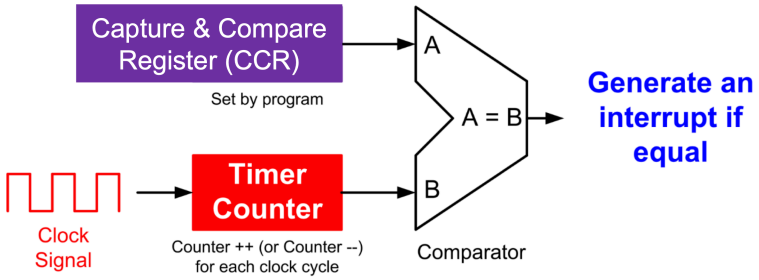
STM32 Timers

- basic block diagram
  - prescaler: divide counter clock frequency by any factor between 1~65536
  - counter: counts from 0 to the value stored in ARR register, then restarts from 0 and generates an overflow event
- timer register
  - TIMx_CNT (16-bit counter)
  - TIMx_ARR (16-bit auto-reload register)
  - TIMx_PSC (16-bit prescaler value)

    $f_{CK\_CNT} = \frac{f_{CK\_PSC}}{TIMx\_PSC+1}, T = \frac{TIMx\_ARR+1}{f_{CK\_CNT}} = \frac{(TIMx\_ARR+1)\times(TIMx\_PSC+1)}{f_{CK\_PSC}}, f_{CK\_OV} = \frac{1}{T}$
  - TIMx_CR1 (16-bit control register, but the most significant 6 bits are reserved)
  - TIMx_SR (16-bit status register, but totally 6 bits are reserved)
  - TIMx_DIER (16-bit DMA/Interrupt enable register, but totally 4 bits are reserved)
- counting mode
  - counting up
  - counting down
  - counting up and down



Output Compare

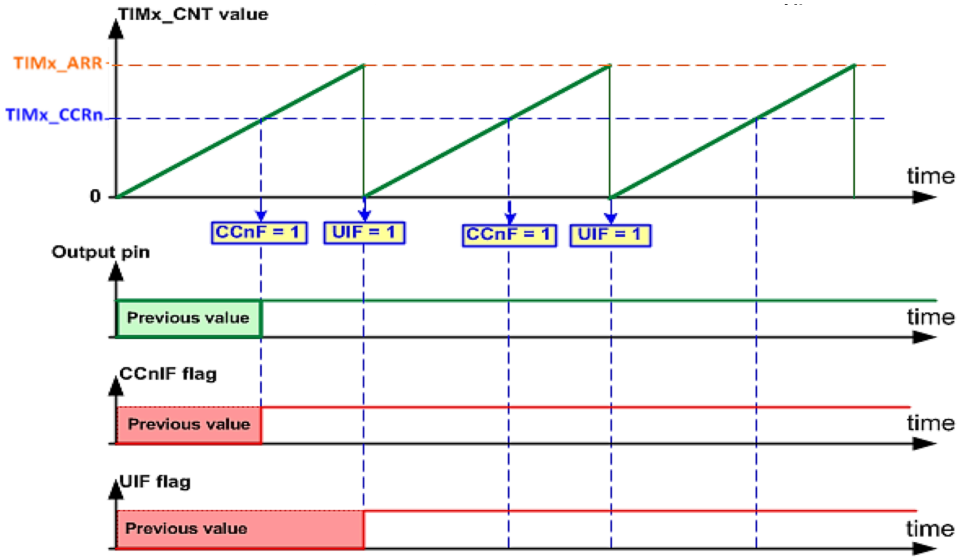


Output Compare Mode (OCnM)	Timer Output (OCREF)
000	Frozen
001	High if CNT == CCR
010	Low if CNT == CCR
011	Toggle if CNT == CCR
100	Forced low (always low)
101	Forced high (always high)
110	PWM Mode 1
111	PWM Mode 2

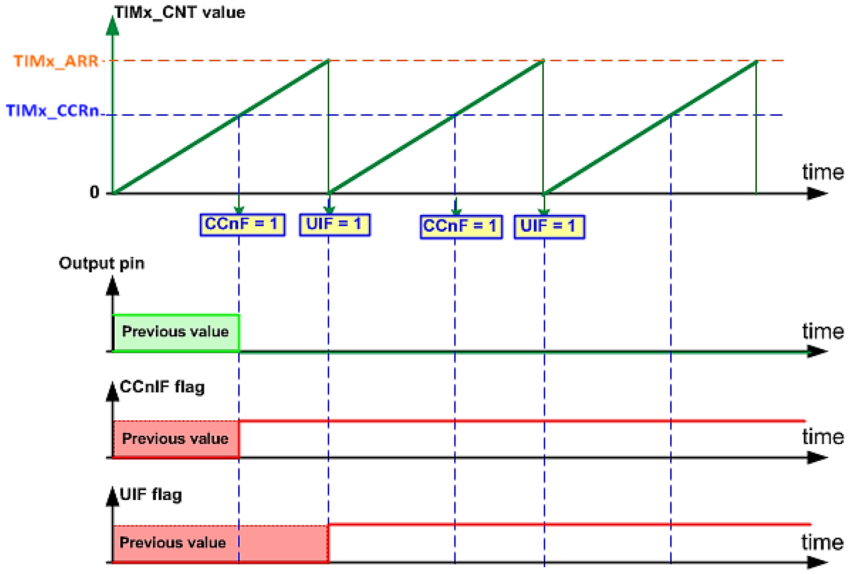
UIF: update interrupt flag

CCnIF: capture & compare channel n interrupt flag

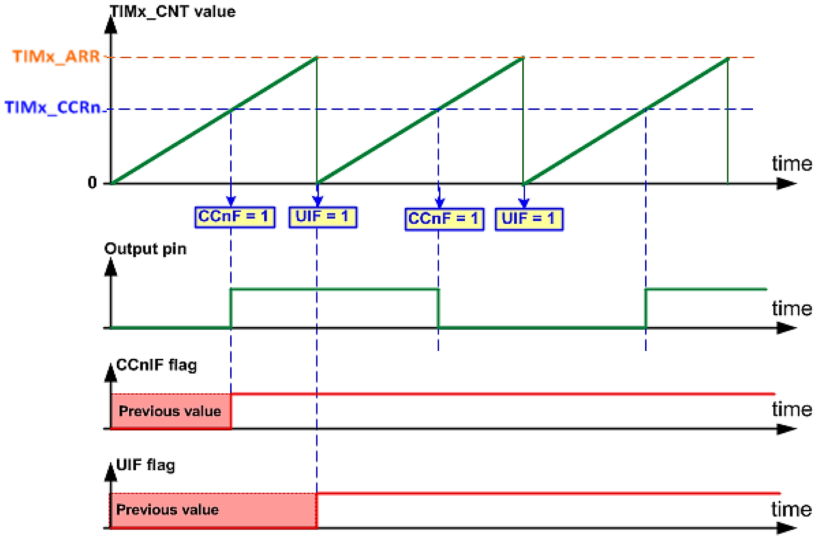
- set mode (OCnM = 001)



- clear mode (OCnM = 010)

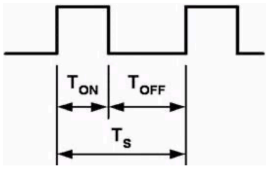


- toggle mode (OCnM = 011)



PWM

- $frequency = 1/T_s = \frac{f_{CK\_PSC}}{(TIMx\_ARR+1)\times(TIMx\_PSC+1)}$



- $duty\ cycle = T_{on}/T_s = T_{on}/(T_{on}+T_{off})$

  $duty\ cycle = \frac{TIMx\_CCR}{TIMx\_ARR+1},\ PW1\ mode$

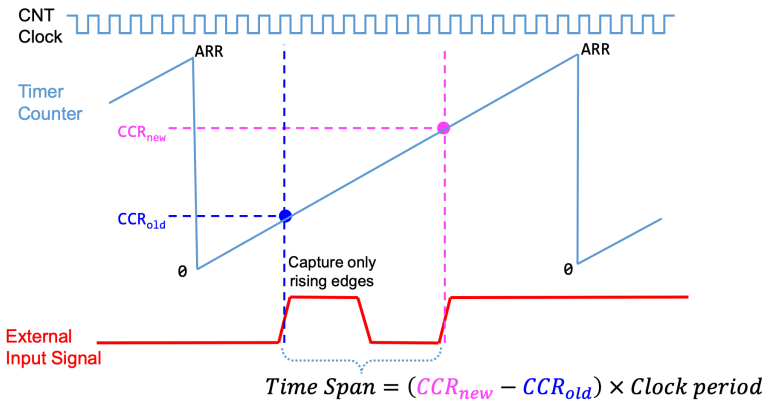
  $= 1 - \frac{TIMx\_CCR}{TIMx\_ARR+1},\ PW2\ mode$

- mode

Mode	Counter < CCR	Counter ≥ CCR
PWM1 mode (Low True)	High	Low
PWM2 mode (High True)	Low	High

- $period = (1 + TIMx\_ARR) \times T_s$

Time Span



$Time\ Span = (CCR_{new} - CCR_{old}) \times Clock\ period$

System Timer (SysTick)

- internel interrupt
- 24-bit downcounter from RELOAD value to 0
- SysTick_LOAD: SysTick reload value register (similar to TIMx_ARR)
- SysTick_VAL: SysTick current value register, has random value on reset (similar to TIMx_CNT)
- time interval between two SysTick interrupts = (RELOAD + 1) * SysTick_Clock_Period

---

Lecture 9: I2C and SPI

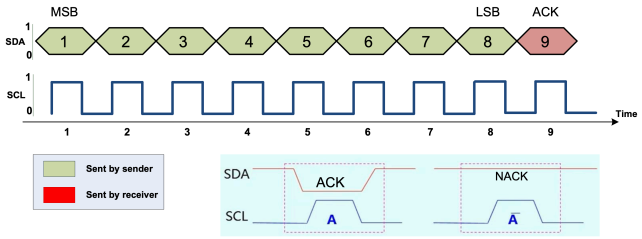
I2C (Inter Integrated Circuit)

- characteristics
  - inter integrated circuit
  - two bidirectional open-drain lines, plus ground (Serial Data Line + Serial Clock Line)
  - serial
  - half-duplex
  - synchrnous
  - byte-oriented, MSB first, (8 bits + 1 ack bit) per data frame
  - multi-master, multi-slave
  - no global master to generate clock, i.e., each master drives both SCL and SDL
- communication steps
  - master sends start condition: SDA changes from 1 to 0 when SCL = 1
  - master sends address of the slave and the direction (is also data), and then receive an ack bit
    - 7-bit address

      each device listens to address and switches state if its address matches
    - 1-bit direction (0: master writes data; 1: master receives data)
  - slave send ack to response master
  - repeat:
    - transmitter sends a byte of data which will be put on SDA

      when SCL = 0, transmitter sends a bit of data onto SDA (SDA allowed to change)

      when SCL = 1, receiver reads a bit of data from SDA (SDA must remain stable)
    - receiver sends ack (ACK - pull down; NACK - leave high)



  - master sends stop condition: SDA changes from 0 to 1 when SCL = 1
- master vs slave
  - each device can be master or slave
  - master should do
    - begins the communication
    - chooses the slave
    - makes the clock
    - sends/receives data

    - slave should do
        - has a unique 7-bit address
        - responds to the master
- working modes

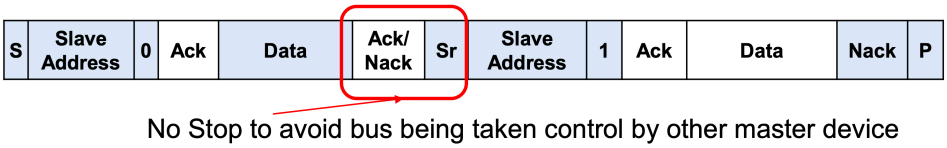
  blue blocks: data transmitted from the master to the slave

  white blocks: data transmitted from the slave to the master

    - master send data to slave

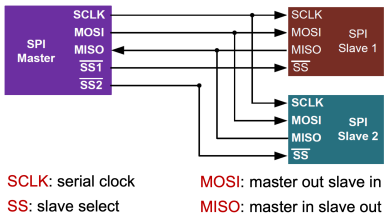
    - slave sends data to master

    - repeated start

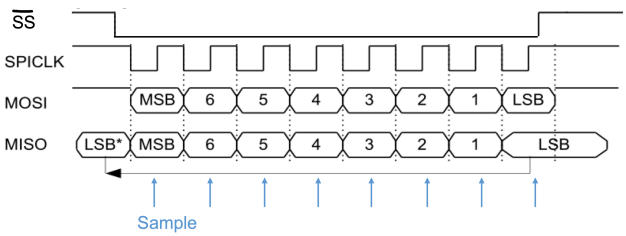


SPI (Serial Peripheral Interface)

- characteristics
    - serial
    - synchronous
    - full-duplex
    - single-master, multi-slave
    - no start/stop or slave ack, master sets corresponding slave select (SS) signal instead

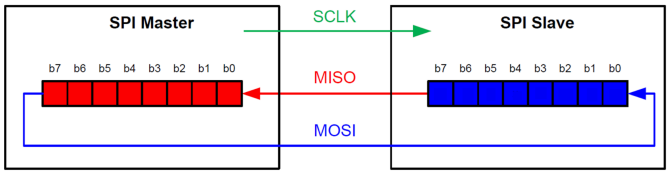


- SPI timing diagram
    - MSB first (but can be set to LSB first)
    - transmit data at rising/falling edge of SPICLK; the same bit data is read at the following falling/rising edge

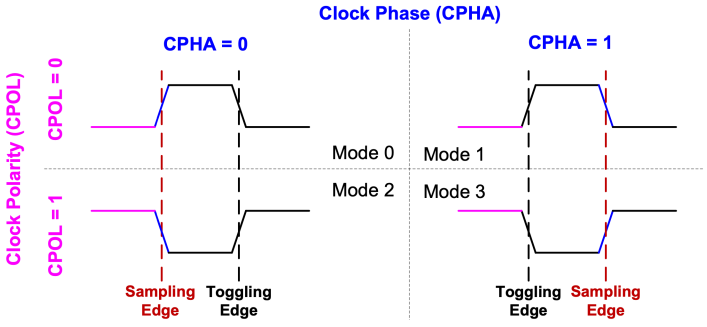


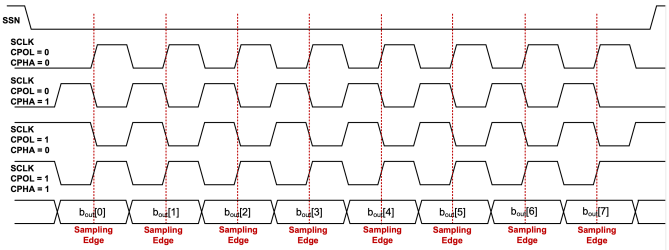
    - if the master only write to the slave, it can ignore data received from the slave

if the master needs to read from the slave, it should send an empty data to trigger the slave to send data



- SPI clock phase (CPHA) & polarity (CPOL)
  - polarity: whether the clk signal is high or low when it is idle
    - 0: sclk = 0 when idle
    - 1: sclk = 1 when idle
  - phase: whether the data is sampled on the rising or falling edge of clk
    - 0: sample at 1st edge
    - 1: sample at 2nd edge
  - CPOL and CPHA together determine the clock edge for transmitting and receiving
  - SCLK is pushed to low during idle when CPOL = 0, otherwise pulled to high during idle





CPOL	CPHA	Data Read and Change Time	SPI Mode
0	0	Read on rising edge, changed on a falling edge	0
0	1	Read on falling edge, changed on a rising edge	1
1	0	Read on falling edge, changed on a rising edge	2
1	1	Read on rising edge, changed on a falling edge	3

I2C vs SPI

- commonality
  - both serial and synchronous
  - both for short distance communication
  - similar application scenarios
  - both in master-slave configuration
- difference
  - I2C by Philips while SPI by Motorola

- I2C is half-duplex while SPI is full-duplex
- I2C uses ack but SPI doesn't
- I2C addresses
- devices by broadcasting slave addresses onto the bus while SPI addresses devices by sending an chip select signal to the corresponding slave
- I2C has fixed clock polarity and clock phase while SPI allows adjustable clock polarity and clock phase

---

Lecture 10: Bus

Definition

- a shared communication link between subsystems
- multiple components can connect to the bus and exchange data through set of wires
- each wire of a bus can contain a single binary digit at any time
- at a given time slot, only one component can send data to the bus

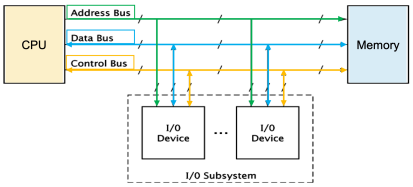
Classification

- address bus (unidirectional)

  CPU accesses memory by addressing a unique location
- data bus (bidirectional)

  carry data between src and dst
- control bus (bidirectional)

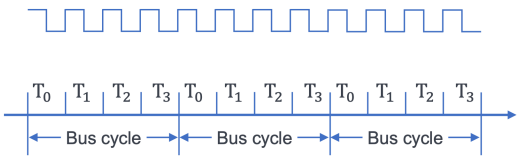
  containing signal requests and acks which indicate what type of information is on the data lines



Performance

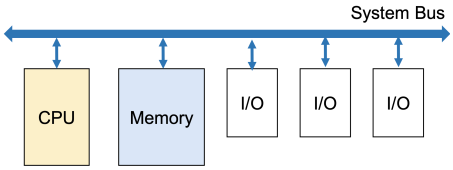
- width: the max number of data bits transmitted simultaneously (e.g. 32-bit)
- bandwidth (i.e., transmission rate): the max amount of data transmitted on a bus within a specific time (e.g. MB/s)
- frequency: speed at which the bus operates
- bandwidth = width * frequency

  note that: width is represented in bit but not byte; the frequency is not the bus clk frequency

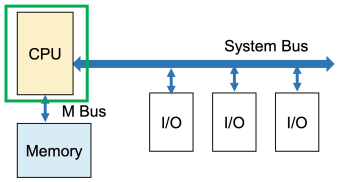


Architecture

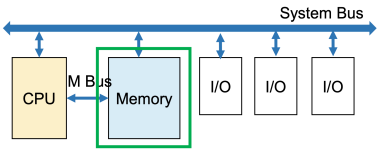
- single level bus
  - one common bus is used to communicate between peripherals and microprocessors
  - easy to add or remove IO devices
  - conflicts
  - occur when multiple components need to use the bus, and priority needs to be determined
  - not support burst transfer (which refers to the consecutive transfer of multiple data elements, initiated by a single address, i.e., the data flow looks like: address, data, data, data, ...)



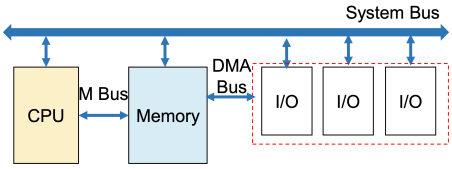
- two level bus
  - CPU centralized: easy to add or remove IO devices



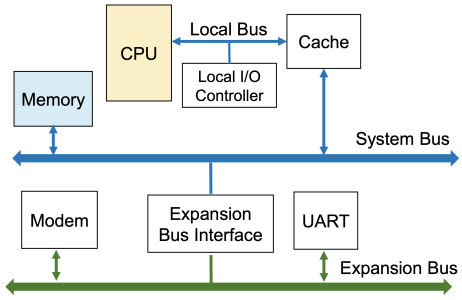
  - Memory centralized: Memory bus (M-bus) operates at a high speed, reducing the burden on the system bus



- multi-level bus
  - Memory centralized with DMA: information exchange between IO devices and main memory doesn't require CPU intervention



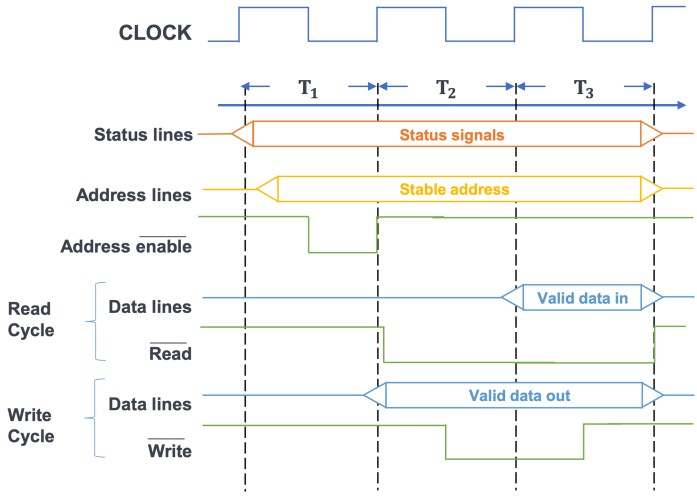
  - more advanced: allowing various components to connect efficiently while maintaining high performance and data integrity



Timing

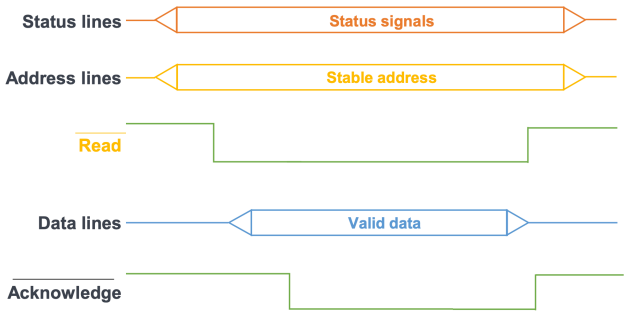
- bus transaction step
  - request: a device that wishes to initiate a transaction on the bus sends a request signal to the bus controller or arbiter
  - addressing: master sends address and control signal to the bus
  - transfer: master and slave perform data trainsmission
  - complete: master removes relevant information from the bus and release the control from the bus
- coordination of events on bus
  - synchronous: controlled by a clock

    e.g. CPU (Master) reads/writes data from/to memory (Slave)

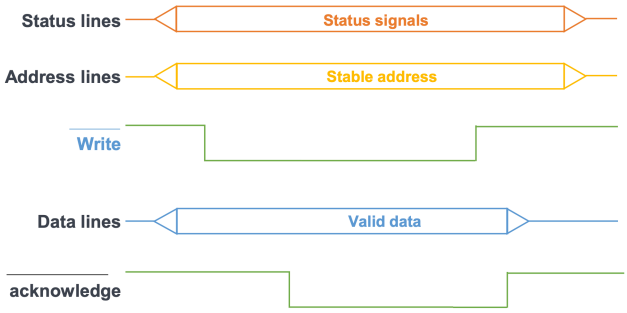


  - asynchronous: handled by well-defined specifications, i.e., a response is delivered within a specified time after a request

    e.g. CPU (master) reads data from memory (slave)

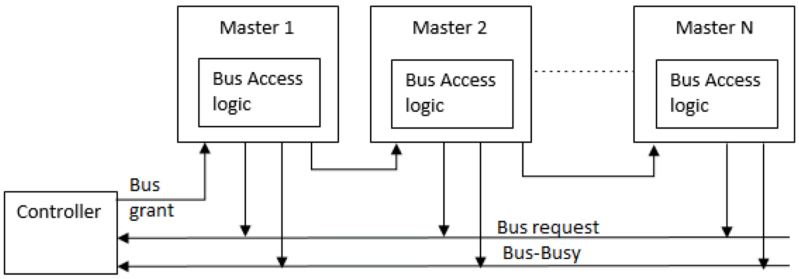


    e.g. CPU (master) writes data to memory (slave)

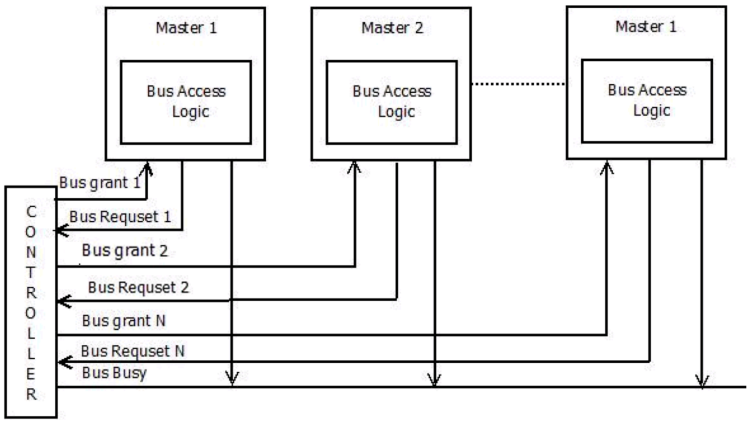


Arbitration

- chooose a mster among multiple requests
- centralized: single hardware device controls bus access, which may be part of CPU or separate
  - serial arbitration (daisy chain): devices connected in a chain sequentially select the bus controller
    - all bus master use the same line for bus request
    - the bus controller gives the bus grant signal if the bus busy line is inactive
    - bus grant signal is propagated serially through all masters starting from nearest one

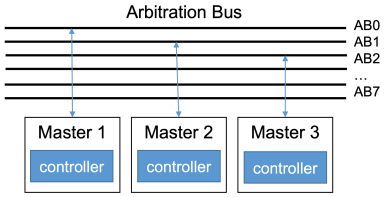


  - parallel arbitration (independent request): devices independently request bus access
    - all bus masters have individual request and grant lines
    - the bus controller knows which master has requested so bus is granted to that master
    - priorities of the masters are predefined so on simultaneous bus requests, the bus is granted based on the priority



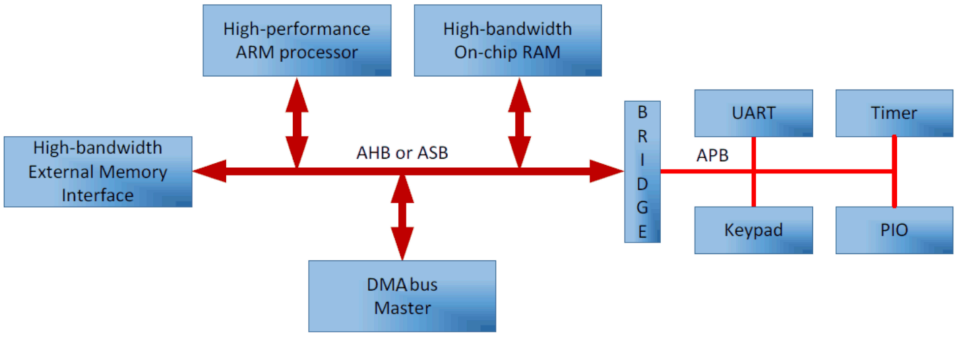
- distributed: there are control logics (arbiters) on all modules and each module may claim the bus
  - the bus device arbiter sends its own arbitration number onto the shared bus, compares the arbitration number received from bus with its own number to determine priority, and withdraws its arbitration number if its own priority is lower
  - finally the highest-priority arbiter remaining on the bus gains control of the bus

- one device failure won't affect others



SoC AMBA bus

- advantage
  - enable IP reuse
  - flexibility
  - compatibility
  - well-supported



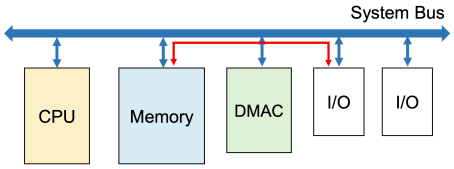
- AHB (Advanced High-performance Bus)
  - high performance
  - connects CPUs, High-Speed Memory, DMA Controllers
  - high bandwidth data transfer
  - multiple masters supported
  - ideal for high-performance components
- APB (Advanced Peripheral Bus)
  - low power
  - connects low-speed peripheral devices to system bus
  - suitable for Timers, GPIO Controllers, UART, etc.

Lecture 11: DMA

Transfer Direction

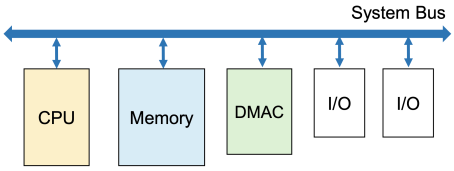
- external device to memory
- memory to external device
- memory to memory
- external device to another external device (potentially)

no continuous intervention by the processor, use DMAC (an peripheral control unit of DMA) to move data

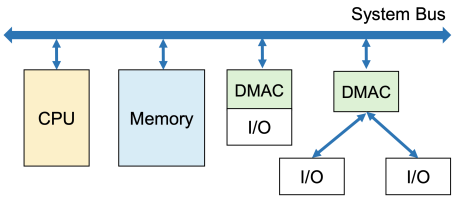


Configuration

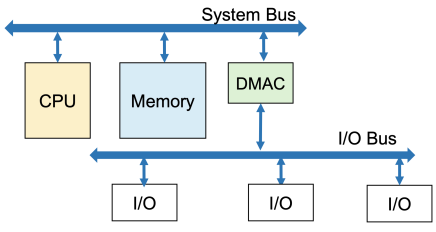
- single bus, detached DMA controller
  - each transfer uses bus twice (IO to DMAC and then DMAC to memory)
  - CPU is suspended twice



- single bus, integrated DMA controller
  - controller may support more than one device
  - each transfer uses bus once (DMAC to memory)
  - CPU is suspended once

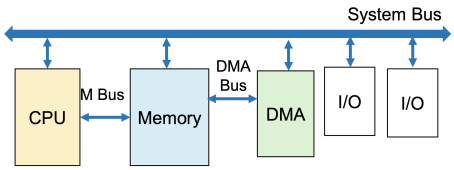


- separate IO bus
  - bus supports all DMA enabled devices
  - each transfer uses bus once (DMAC to memory)
  - CPU is suspended once



- separate DMA bus
  - without CPU suspended

- conflict when both CPU and DMA access memory



Transfer Mode

- burst mode

  entire block of data transferred in one continuous operation, during which CPU remains inactive, ensuring high efficiency

- interleaving mode

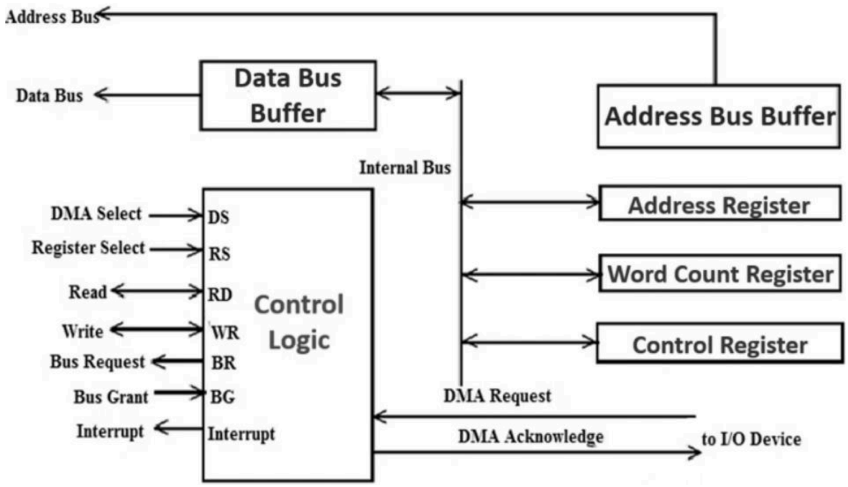
  data transferred only when there is no conflict with CPU for system bus or memory access, ensuring no interference

- cycle stealing mode

  firstly, DMA controller takes control; once one word of data transferred, DMAC releases control back to the CPU, and then repeat this process, ensuring that CPU can continue exexuting its instructions between data transfers

DMAC Registers

- address register: contains the address to specify the desired location in memory
- word count register: contains the number of words to be transferred
- control register: specifies the transfer mode



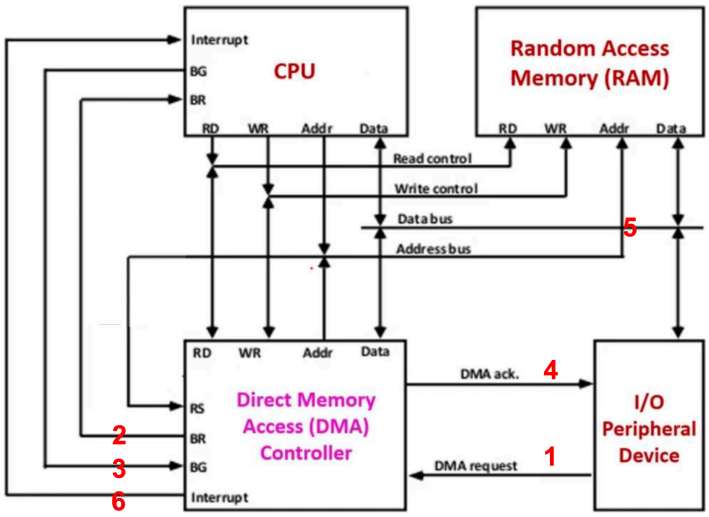
Operation

- interaction with CPU
  - CPU sets up registers configuration
    - data transfer direction between memory and IO (Read/Write)
    - IO devices address
    - starting address of memory block for data

        - amount of data to be transferred
    - CPU carries on with other work
    - DMAC deals with transfer
    - DMAC send interrupt when finished
- interaction with peripheral
    - when each word of data is available, the IO device places a signal on the DMA request wire
    - the signal causes DMAC to seize the memory bus, in order to place the desired address on the memory address wire and a signal on the DMA acknowledge wire
    - when IO device receives the DMA acknowledge signal, the transfer between external device and DMA starts

Sequence

- I/O device sends a DMA request to DMAC
- DMAC sends a bus request signal to CPU for memory access
- CPU sends bus grant signal to DMAC, and release the bus control authority to DMAC
- DMAC sends acknowledgement signal to I/ O device
- DMAC transfers data between I/O device and Memory according to preset address and word count
- After completing of transfer, DMAC cancels the bus request signal
- DMAC sends interrupt signal to CPU if necessary

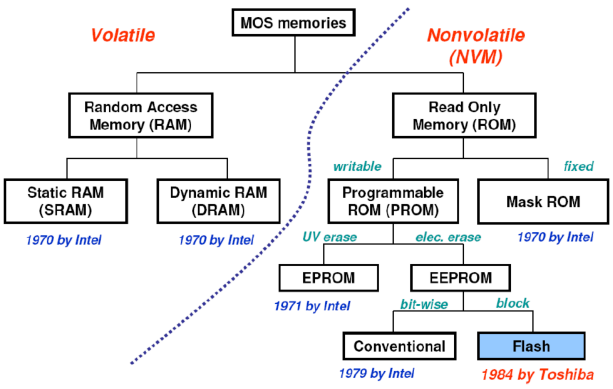


Lecture 12: Embedded Storage Management

Type

- volatile: loses its content when power-off
  - SRAM (Static Random Access Memory)
    - low density
    - high power
    - expensive but fast
    - content last until power-off
    - often used for cache
  - DRAM (Dynamic Random Access Memory)
    - high density
    - low power
    - cheap but slow
    - refresh regularly
    - often used for main memory
- non-volatile: remain its content even when power-off
  - ROM (Read-Only Memory)
    - permanent, pre-programmed data
    - retains information across power cycles
  - EPROM/Flash (Erasable Programmable ROM)
    - rewritable
    - often used used for firmware, configuration and data storage
    - EPROM can be erased using UV light, while EEPROM can be erased electrically

Note that erasing means changing from 0 to 1, while writing means from 1 to 0



Flash

- sector
  - refers to a fixed-size, contiguous block of storage
  - the smallest addressable unit for reading/writing data and the smallest erasable unit
- interface: SPI
- application
  - SD Card
  - USB Flash Drive
  - SSD

SD Card

- interface
  - SD mode
    - clk
    - 4 data lines
  - SPI mode
    - clk
    - card select
    - 2 data lines

Pin	SD	SPI
1	DAT3	CS
2	CMD	DI(MOSI)
3	Vss	Vss
4	Vcc	Vcc
5	CLK	CLK
6	Vss2	Vss2
7	DAT0	DO(MOSO)
8	DAT1	Reserved
9	DAT2	Reserved

- driver: Read, Write, Initial, GetStatus

FAT File System

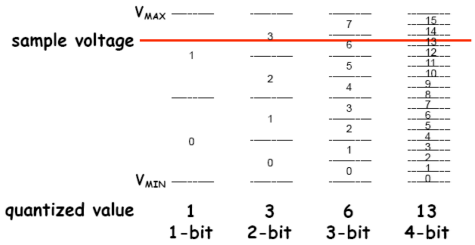
- type
  - FAT
  - NTFS
  - ext2, ext3, ext4
  - CDFS
- cluster
  - 1 cluster = 2^n sectors (e.g. FAT 12: 12-bit cluster address)
  - $File\ System\ Size = num_{clusters} \times size_{cluster} = 2^{cluster\ address\ width} \times size_{cluster}$

	FAT12	FAT16	FAT32
Cluster address width	12 bits	16 bits	28 bits 4 bits reserved
Number of Clusters	2^{12} (4K)	2^{16} (64K)	2^{28} (256M)

Lecture 13: ADC

Sampling

- rate: how often analog signal is measured (samples per second, Hz)
- resolution: number of bits to represent each sample (bit depth, bit)



Output

for a n-bit ADC:

- $numOfSteps = 2^n, stepSize = \frac{V_{ref}}{numOfSteps}$
- $Output = round(\frac{V_{input}}{stepSize}) = round(2^n \times \frac{V_{input}}{V_{ref}})$

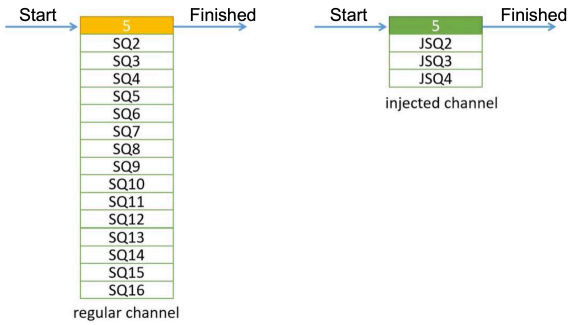
Architecture

- Successive-approximation ADC (SAR): use binary search to determine the closest digital representation of the input signal
- Flash ADC: directly convert an analog input signal into a digital output by using a set of comparators to rapidly compare the input against multiple reference voltages

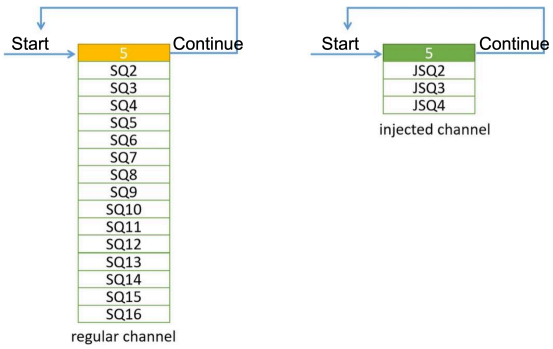
	Resolution	Sampling rate	Cost
SAR ADC	High	Low	Low
Flash ADC	Low	High	High

STM32 ADC

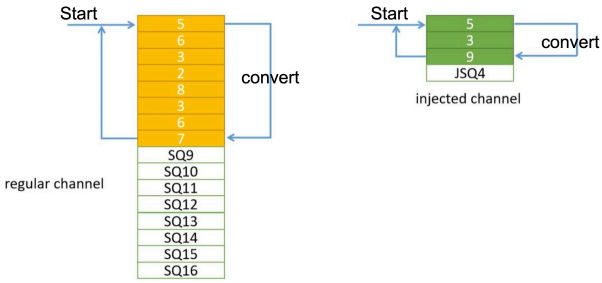
- 12-bit
- 16-channel regular grpup and 4-channel injected group (higher priority)
- single conversion mode



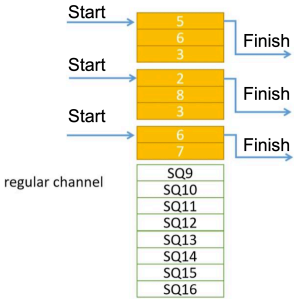
- continuous conversion mode



- continued scan mode



- discontinuous mode



- $t_{ADC} = t_{sampling} + t_{conversion}$
  - $t_{sampling}$ is defined in ADCx_SMPR register (1.5 ~ 239.5)
  - $t_{conversion} = 12.5 \times t_{ADC\ clk\ cycle}$

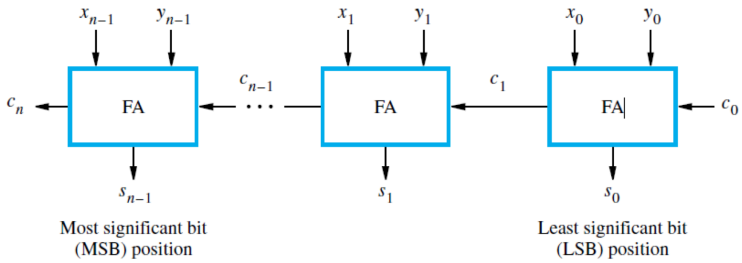
Lecture 14: Arithmetic

Barrel Shifter

- LSL (logical shift left): bits are shifted to the right and the new bits added in at the left hand side are 0
- LSR (logical shift right): bits are shifted to the left and the new bits added in at the right hand side are 0
- ASR (arithmetic shift right): bits are shifted to the right and the new bits added in at the left hand side are 0 for positive value or 1 for negative value
- ROR (rotate right): bits are rotated rightwards so that the bits shifted out at the right hand side reappear at the left hand side
- RRX (rotate extended): bits are shifted right one place only and the carry flag is shifted into the new most significant bit. The least significant bit is shifted into the carry flag only if the mnemonic specifies an S
- shifts with other instruction
  - `mov r0, r0, LSL #n`: $r0 = r0 * 2^n$
  - `mov r0, r0, LSR #n`: $r0 = r0/2^n$ (unsigned)
  - `mov r0, r0, ASR #n`: $r0 = r0/2^n$ (signed)
  - `mov r0, r0, ROR #n`: swap the first 32-n bits and the other n bits
  - `add r0, r0, r0, LSL #n`: $r0 = r0 + r0 * 2^n = r0 * (1 + 2^n)$
  - `rsb r0, r0, r0, LSL #n`: $r0 = r0 * 2^n - r0 = r0 * (2^n - 1)$

ALU Adder

- ripple carry adder



  each cell causes a propagation delay

- lookahead carry adder

  Noticing c_{i+1} is 1 IFF at least two bits among $x_i,\ y_i$ and c_i are 1, we derive that $c_{i+1} = x_iy_i + y_ic_i + c_ix_i = x_iy_i + c_i(x_i \oplus y_i)$. What's more, $s_i = x_i \oplus y_i \oplus c_{i-1}$, so we can define $g_i = x_iy_i,\ p_i = x_i \oplus y_i$, and then $c_{i+1} = g_i + p_ic_i = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1}) = \ldots$ till c_0 occurs.

$$\begin{cases} c_1 = G_0 + c_0P_0 \\ c_2 = G_1 + c_1P_1 = G_1 + (G_0 + c_0P_0)P_1 = G_1 + G_0P_1 + c_0P_1P_0 \\ c_3 = G_2 + c_2P_2 = G_2 + (G_1 + G_0P_1 + c_0P_1P_0)P_2 = G_2 + G_1P_2 + G_0P_2P_1 + c_0P_2P_1P_0 \\ c_4 = G_3 + c_3P_3 = G_3 + (G_2 + G_1P_2 + G_0P_2P_1 + c_0P_2P_1P_0)P_3 = G_3 + G_2P_3 + G_1P_3P_2 + G_0P_3P_2P_1 + c_0P_3P_2P_1P_0 \end{cases}$$

- select carry adder

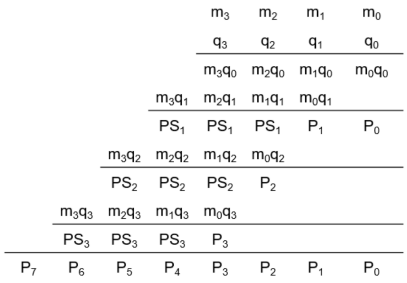
  When calculating $s_i = x_i + y_i + c_i$, even if c_i is unknown till now, we still know that c_i must be 0 or 1. We can use a adder which produces two results, $x_i + y_i$ and $x_i + y_i + 1$, which should be selected referring to c_i when it is known.

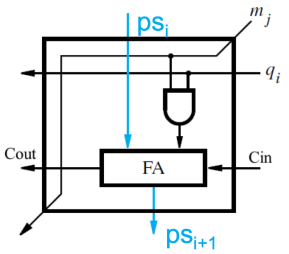
- propagation delays comparison

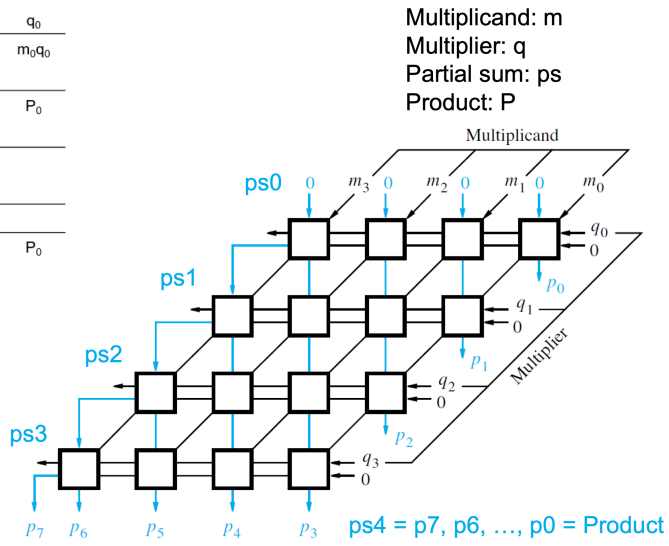
Size of adder	Ripple carry	Look ahead	Carry select
4 bits	4	1	1
8 bits	8	2	1
16 bits	16	4	2
32 bits	32	8	3
64 bits	64	16	4

Multiplier

- matrix multiplier
  - Add each partial product into a total as it is formed, with carry save adder involved. Carry-save addition passes (saves) the carries to the output, rather than propagating them. With this technique, we can avoid carry propagation until final addition.
  - When adding sets of numbers, carry-save can be used on all but the final sum. Standard adder (carry propagate) is used for final sum.
  - Carry-save is fast (no carry propagation) and inexpensive (full adders)





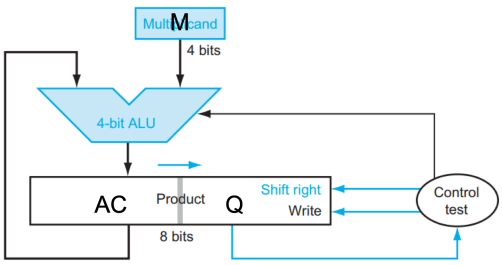


- sequential multiplier

  In each step, one bit of the multiplier is selected. If the bit is logic 1, the multiplicand is shifted left to form a partial product, and it's added to the partial sum.

  Only for unsigned multiplication. Sign bits are evaluated separately.

  Do a n-loop, for each loop, set `AC = Q[0] == 1 ? AC + M : AC`, and then logically shift right {AC, Q}.



- Booth's multiplication algorithm

  Use +1 and -1 to replace continuous 1s, e.g., use `0 +1 0 0 0 −1 0` to represent `30 = 0011110 = 0100000 − 0000010`.

  Can perform negative number multiplication. Sometimes worse than normal algorithm.

Do a n-loop, for each loop, set `{Q[0]Q[-1]} == 01: AC += M; else {Q[0]Q[-1]} == 10: AC += M' + 1; else AC = AC`, and then arithmetically shift right {AC, Q[: -1]}.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

- Booth2 multiplication algorithm

$Q_{i+1}Q_iQ_{i-1}$	Equivalent value (at position i)	Operation
000	0	+0
001	+1	+M
010	+1	+M
011	+2	+2M
100	-2	-2M
101	-1	-M
110	-1	-M
111	0	0

Editted by Ruixiang Jiang.