Part1

Fixed Point, Floating Point and Number Representations

Qm.n => 1一位标志位，m位小数点钱，n位小数点后

Denormalized和Normalized：
http://cenalulu.github.io/linux/about-denormalized-float-number/

Dynamic range = $20\log_{10}(\frac{\text{largest number}}{\text{smallest positive number}})$
Precision = difference between 2 consecutive numbers

IEEE 754

- Sign(1bit) Exp(8bit) Mantissa(23bit)
- Normalized value = $(-1)^s \times 2^{Exp-127} \times M$
  M = 1.Mantissa
- Denormalized value = $(-1)^s \times 2^{-126} \times M$
  M = 0.Mantissa

注意IEEE 754浮点数的最大值，最小值，最小正数
有的部分有保留字，最大值和最小值不是全1和全0
https://zh.wikipedia.org/wiki/IEEE_754#%E7%89%B9%E6%AE%8A%E5%80%BC

最小的规约数 指数域：0000 0001
最大的规约数 指数域：1111 1110
非规约数 指数域：0000 0000

方框T的意思是延时，T means sampling interval，同样意思的还有z^{-1}

Quantization, Truncation and Round-off

- mid-tread
  Number of quantizer levels = $2^n – 1$
  Quantization step size $Q = \frac{V_{\max} – V_{\min}}{2^n – 1}$
  Output values ... –Q, 0, Q, 2Q, 3Q, ...
- mid-rise Number of quantizer levels = 2^n
  Quantization step size $Q = \frac{V_{\max} – V_{\min}}{2^n}$
  Output values ... –0.5Q, 0.5Q, 1.5Q, 2.5Q, 3.5Q, ...

Quantization error $e(n)$
mean = 0
variance = $\frac{Q^2}{12}$
Auto-correlation $R(m) = \sigma^2 \delta(m)$
power spectral density $S(e^{j\omega}) = \sigma^2$

Signal variance = σ_x^2

SQNR $= 10\ \log(\dfrac{\sigma_x^2}{\sigma_e^2}) = 10\ \log(\dfrac{3\ \sigma_x^2\ 2^{2n}}{V_{\max}^2}) = 6.02n + 4.77 + 20\ \log \frac{\sigma}{V_{\max}}$

Part1 P61

- Rounding:
  mean = 0
  variance = $\frac{Q^2}{12}$
- Truncation:
  mean = $-\frac{Q}{2}$
  variance = $\frac{Q^2}{12}$

e(n) --> h(n) --> v(n)
$\eta_v = H(e^{j0})\eta_e$

$\sigma_v^2 = ||H(e^{j\omega})||_2^2\ \sigma_e^2$

$||H(e^{j\omega})||_2^2 = \int_{-\pi}^{\pi} |H(e^{j\omega})|^2 \frac{d\omega}{2\pi}$

$S_{\text{out}}(e^{j\omega}) = |H(e^{j\omega})|^2 S_{\text{in}}(e^{j\omega})$

Part1 P83, 89

Part2

Sampling

Part 2 P10

$x_c(t)$ is centered around some non-zero frequency f_c

$\frac{2f_c + B}{m+1} \le f_s \le \frac{2f_c - B}{m}$
$m = 0, 1, 2, 3 \ldots$

Types of adc PART 2 P45

Real-Time Operations: Scheduling

Latency = 7 clocks 包含首尾
Part 2 P114, 116

Part3

Adder

Half Adder Part3 P4
Full Adder:

- Ripple Carry Adder:
  $s_n = a_n \oplus b_n \oplus c_{n-1}$
  $c_n = a_n b_n + b_n c_{n-1} + a_n c_{n-1}$
  s_n 延迟1
  c_n 延迟2
- Carry Look-Ahead Adder:
  Part3 P11

Multiplier

整个Multiplier都不考

/2 --> 右移一位，左边填充符号位 sign extension
*2 --> 左移一位，右边补0

Booth Multiplier Part3 P42
Wallace Tree Multiplier Part3 P49

Part4

Algorithm strength reduction

Loop bound of a loop $= \dfrac{\text{loop computation time}}{\text{no of delays in the loop}}$

Critical path = longest computation time among all zero delay paths
Iteration bound = maximum loop bound
Critical loop = the loop with maximum loop bound

Latch = D = T

Sampling period $T_s \geq \text{critical path}$

Pipelining

P12

Retiming

Part 4 P19

Pipelined recursive filters

Part 4 P31

Unfolding

Part5

Real-time processing on a DSP processor

Collect N samples at a time
$T_p < NT_s$
Maximum block delay = $2NT_s$

MIPS = number of million MAC (multiply-accumulate) instructions per second
TMS320C55x example: 120 MHz clock, 1 cycle/MAC (when fully pipelined), 2 parallel multiply-accumulator = 240 MIPS

TMS320C55x Programming

AC0 - AC3 40bits
HI(AC0) 16bits(31-16)
LO(AC0) 16bits(15-0)

AR0 - AR7 16bits
XAR0 23bits

T0 - T3 16bits

Data Memory 16bits

MOV can be x bits → y bits, where
x < y
x bits sign extended to y bits, except when x = 23 bits, x bits zero filled to y bits

MOV #11, AC0
MOV #bh, AC0

- Absolute addressing
  MOV *(#11), AC0
- Direct addressing
  MOV @Daddr, AC0
  Daddr, DP 16bits
  DPH 23 - 17 bits of DPX
  address 23bits
  Doffset = 7 lsb bits of (Daddr – DP)
  address bits 22 - 16 = DPH
  address bits 15 - 0 = DP + Doffset
- Auxiliary register (AR) indirect addressing
  MOV AC0, high_byte(*AR0)
  move bits 7-0 of AC0 to bits 15-8 of location at XAR0

MOV *AR0, *AR1, AC0
Move 16-bit data at XAR0 to bits 15-0 of AC0.
Move 16-bit data at XAR1, sign extended to 24-bit, to bits 39-16 of AC0.
MOV AC0, *AR0, *AR1
Move bits 15-0 of AC0 to location at XAR0.
Move bits 31-16 of AC0 to location at XAR1.
MOV dbl(*AR0), dbl(*AR1)
move 2 consecutive data

*ARx± increase/decrease ARx after addressing
*±ARx increase/decrease ARx before addressing

*ARx(offset)
no modification to ARx, but offset is used while addressing. offset = AR0, Ty, #K16

MOV *(AR0+T0), AR1