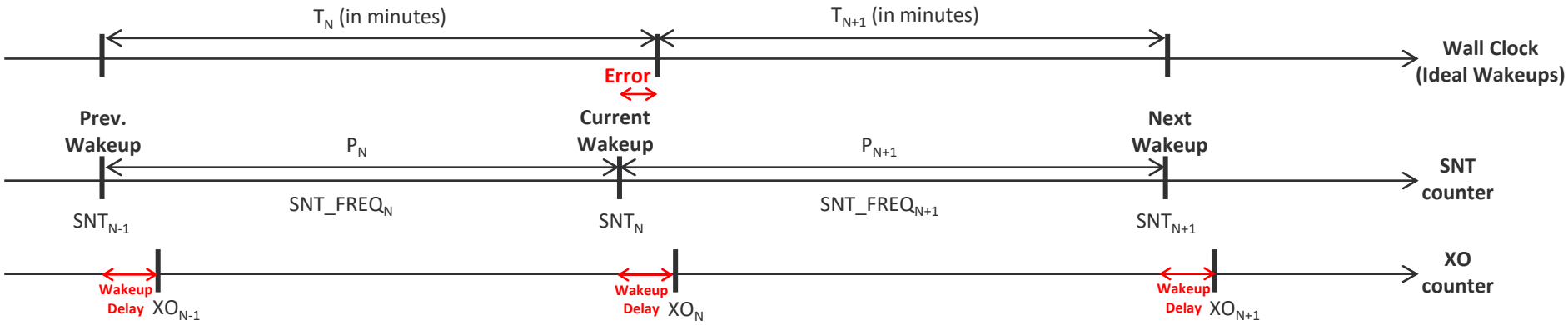
SNT Calibration



- Current SNT Threshold
  - $SNT_N = SNT_{N-1} + P_N$
- Frequency Adjustment
  - $SNT\_FREQ_{N+1} = \frac{P_N}{Actual\ Elapsed\ Time} = \frac{P_N \times XO\_FREQ}{XO_N - XO_{N-1}} = SNT\_FREQ_N \times \frac{P_N \times XO\_FREQ}{SNT\_FREQ_N (XO_N - XO_{N-1})}$
- Interval Adjustment
  - $Error(in\ sec) = Expected\ Elapsed\ Time\ - Actual\ Elapsed\ Time = \frac{P_N}{SNT\_FREQ_N} - \frac{XO_N - XO_{N-1}}{XO\_FREQ}$
- Calculate P_{N+1}
  - $P_{N+1} = 60 \times T_{N+1} \times SNT\_FREQ_{N+1} + Error(in\ sec) \times SNT\_FREQ_{N+1}$
    $= 60 \times T_{N+1} \times SNT\_FREQ_{N+1} + \left(\frac{P_N}{SNT\_FREQ_N} - \frac{XO_N - XO_{N-1}}{XO\_FREQ}\right) \times \frac{XO\_FREQ \times P_N}{XO_N - XO_{N-1}}$
    $= 60 \times T_{N+1} \times SNT\_FREQ_{N+1} + P_N \left(\frac{P_N \times XO\_FREQ}{SNT\_FREQ_N (XO_N - XO_{N-1})} - 1\right)$
- Next SNT Threshold
  - $SNT_{N+1} = SNT_N + P_{N+1}$

SNT Calibration

- Equations (All terms are positive integers)
  - $SNT\_FREQ_{N+1} = SNT\_FREQ_N \times \frac{P_N \times XO\_FREQ}{SNT\_FREQ_N(XO_N - XO_{N-1})}$
  - $P_{N+1} = 60 \times T_{N+1} \times SNT\_FREQ_{N+1} + P_N \left(\frac{P_N \times XO\_FREQ}{SNT\_FREQ_N(XO_N - XO_{N-1})} - 1\right)$
  - $SNT_{N+1} = SNT_N + P_{N+1}$

- How to calculate
  - $\frac{P_N \times XO\_FREQ}{SNT\_FREQ_N(XO_N - XO_{N-1})}$: Needs to be represented as a decimal number. Its value is very close to 1.
    - Use a custom decimal number representation (see the last slide)
  - $P_N(A-1)$, where $A = \frac{P_N \times XO\_FREQ}{SNT\_FREQ_N(XO_N - XO_{N-1})}$ using the decimal number representation
    - If $A \geq 1$
      - $P_N(A-1) = P_N \times (A \,\&\, 0xBFFFFFFF)$
    - If $A < 1$ (i.e., $A[31:30] = 2'b00$)
      - Calculate $P_N(1-A)$ and subtract this from $(60 \times T_{N+1} \times SNT\_FREQ_{N+1})$ when calculating P_{N+1}
      - $P_N(1-A) = P_N\left(1 - \sum_{i=0}^{29} \frac{n_i}{2^{i+1}}\right)$, where n_i is the i-th bit in A
        $= P_N\left(\sum_{i=0}^{29} \frac{1}{2^{i+1}} + \frac{1}{2^{30}} - \sum_{i=0}^{29} \frac{n_i}{2^{i+1}}\right) = P_N\left(\sum_{i=0}^{29} \frac{1-n_i}{2^{i+1}} + \frac{1}{2^{30}}\right)$
        $= P_N \times [(\sim A) \,\&\, 0x3FFFFFFF] + (P_N \gg 30)$

SNT Calibration

- Final Equations, where $A = \frac{P_N \times XO\_FREQ}{SNT\_FREQ_N (XO_N - XO_{N-1})}$ (using the decimal number representation)
  - $SNT\_FREQ_{N+1} = SNT\_FREQ_N \times A$
  - If $A \geq 1$: $P_{N+1} = 60 \times T_{N+1} \times SNT\_FREQ_{N+1} + P_N \times (A \,\&\, 0xBFFFFFFF)$
    If $A < 1$: $P_{N+1} = 60 \times T_{N+1} \times SNT\_FREQ_{N+1} - P_N \times [(\sim A) \,\&\, 0x3FFFFFFF] - (P_N \gg 30)$
  - $SNT_{N+1} = SNT_N + P_{N+1}$
- NOTE:
  - Except A, all terms are positive integers
  - A is very close to 1.
  - '$P_N \times (A \,\&\, 0xBFFFFFFF)$' is a positive integer.
  - '$P_N \times [(\sim A) \,\&\, 0x3FFFFFFF]$' is a positive integer.

div and mult functions

- Simple 32-bit Decimal Number Representation
  - [31:30] – integer part, [29:0] – non-integer part where bit[n] indicates $1/2^{(n+1)}$
  - Example: 1.5 = 32'b 0100 0000 0000 0000 0000 0000 0000 0001
    0.999999999068677 = 32'b 0011 1111 1111 1111 1111 1111 1111 1111

```
// Division
// Input:  dividend (unsigned integer)
//         divisor (unsigned integer)
// Output: result (decimal)
uint32_t divide (uint32_t dividend, uint32_t divisor) {
    uint32_t result = 0;

    // Get the integer part
    while (dividend >= divisor) {
        result += (1 << 30);
        dividend -= divisor;
    }

    // Get the non-integer part
    uint32_t idx = 0;
    divisor = divisor >> 1;
    while ((dividend!=0) && (divisor!=0) && (idx<30)) {
        if (dividend >= divisor) {
            result |= (0x1 << idx);
            dividend -= divisor;
        }
        divisor = divisor >> 1;
        idx++;
    }
    return result;
}
```

```
// Multiplication
// Input:  num_int (unsigned integer)
//         num_dec (decimal)
// Output: result (unsigned integer)
uint32_t mult_dec (uint32_t num_int, uint32_t num_dec) {
    uint32_t int_part = num_dec>>30;
    uint32_t dec_part = num_dec&0x3FFFFFFF;
    uint32_t result = 0;

    // Integer part
    while (int_part > 0) {
        result += num_int;
        int_part--;
    }

    // Non-Integer part
    num_int = num_int >> 1;
    while ((num_int!=0) && (dec_part!=0)) {
        if (dec_part&0x1) {
            result += num_int;
        }
        num_int = num_int >> 1;
        dec_part = dec_part >> 1;
    }
    return result;
}
```