Embedded systems engineering
Distributed real-time systems

David Kendall

Time is Central to Real-time and Embedded Systems

Several timing analysis problems:

- Worst-case execution time (WCET) estimation
- Estimating distribution of execution times
- Threshold property: can you produce a test case that causes a program to violate its deadline?
- Software-in-the-loop simulation: predict execution time of particular program path

ALL involve predicting an execution time property!

WCET and BCET

- Remember the simple formula for response time analysis (no blocking or jitter).

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Each C_i represents the *worst-case computation time* of its task.
- Worst-case computation time (WCET) : the longest time taken by a program code to complete its execution (assuming no blocking, jitter or interference)
- Best-case computation time (BCET) : the shortest time taken by a program code to complete its execution (assuming no blocking, jitter or interference)
- How to obtain values for C_i?

Calculating execution times

- Measurement
  - Need to exercise great care in obtaining measurements
  - Need to take care in interpreting results
  - How to know if you've measured the worst (best) case?
- Analysis
  - Intended to guarantee that the worst (best) case execution time is reported
  - Difficult to take account of all architectural effects: pipelines, caches, speculative execution etc.
- Let's consider a diagram that illustrates the relationship between these approaches

Timing analysis of systems

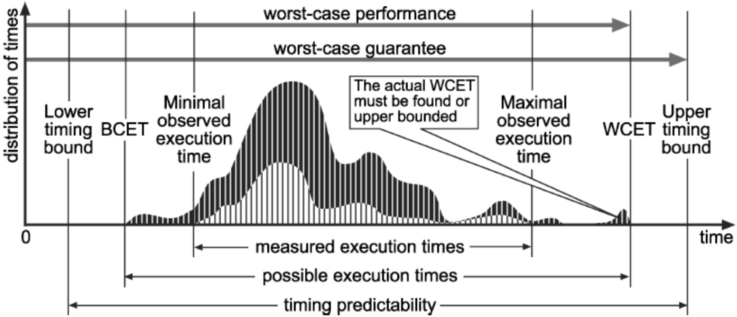


Fig. 1. Basic notions concerning timing analysis of systems. The lower curve represents a subset of measured executions. Its minimum and maximum are the *minimal* and *maximal observed execution times*, respectively. The darker curve, an envelope of the former, represents the times of all executions. Its minimum and maximum are the *best-* and *worst-case execution times*, respectively, abbreviated BCET and WCET.

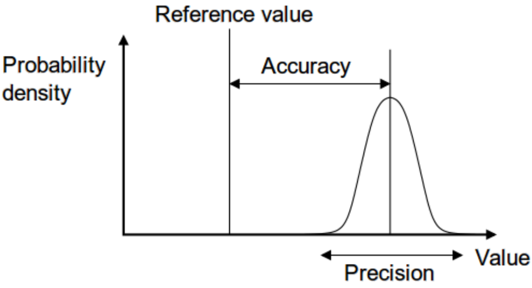
Figure from R.Wilhelm et al., *ACM Transactions on Embedded Computing Systems*, Vol. 7:3, pp 36:1 – 36:53, April 2008.

Measuring execution time

Accuracy, precision, and resolution

- Real-world effects introduce *uncertainty* into measurements.
- Three important features that characterise the quality of measurements are:
  - Accuracy: an indication of the closeness of measurements of a quantity to that quantity's actual value according to some well-defined standard.
  - Precision: the degree to which repeated measurements under unchanged conditions give the same results.
  - Resolution: the smallest change in the underlying physical quantity that produces a noticeable response in the measurement.

Distinguishing accuracy and precision



Accurate but not precise

Precise but not accurate

Figures from http://en.wikipedia.org/wiki/Accuracy_and_precision

Measurement errors

- Difficult to separate individual contributions made to error by accuracy, precision, and resolution
  - Quantifying accuracy is hard, e.g. quantifying accuracy of interval timer requires calibration of clock source with standard measurement of time
  - Use variance of measurements to quantify precision
  - Resolution usually easy to quantify, e.g. resolution of interval timer can introduce an error of ± 1 clock period.
- Other sources of error include:
  - Perturbing the quantity to be measured by the act of measuring it, e.g. program statements added to a program to access a timer change the behaviour of the program being measured
  - Other non-deterministic events may also perturb the quantity to be measured
- Useful to classify errors as either systematic or random

Methods for measuring execution time

Method	*Typical Resolution*	*Typical Accuracy*	*Granulariy*	*Difficulty of Use*
stop-watch	0.01 sec	0.5 sec	program	easy
date	0.02 sec	0.2 sec	program	easy
time	0.02 sec	0.2 sec	program	easy
prof and gprof	10 msec	20 msec	subroutines	moderate
clock()	15-30 msec	15-30 msec	statement	moderate
software analyzers	10 μsec	20 μsec	subroutine	moderate
timer/counter chips	0.5–4 μsec	1-8 μsec	statement	very hard
logic or bus analyzer	50 nsec	half μsec	statement	hard

Figure from David B. Stewart, Measuring execution time and real-time performance (part 1), Dr. Dobbs Journal, November 2006 (available here)

Defining an interval timer for FRDM-K64F

PIT Initialisation

```
void counterInit(void) {
    /* Open the clock gate to the PIT */
    SIM->SCGC6 |= (1u << 23);

    /* Enable the clock for the PIT timers.
       Continue to run in debug mode
     */
    PIT->MCR = 0u;

    /* Disable the timer */
    PIT->CHANNEL[2].TCTRL &= ~PIT_TCTRL_TEN_MASK;

    /* Period p = maximum available, bus clock f = 60 MHz, v = pf - 1 */
    PIT->CHANNEL[2].LDVAL = 0xFFFFFFFF;

    /* Clear the timer interrupt flag */
    PIT->CHANNEL[2].TFLG |= PIT_TFLG_TIF_MASK;

    /* Enable interrupt on timeout */
    PIT->CHANNEL[2].TCTRL |= PIT_TCTRL_TIE_MASK;

    /* Enable the interrupt in the NVIC */
    NVIC_EnableIRQ(PIT2_IRQn);
}
```

Defining an interval timer . . .

```
void counterStart(void) {
    /* Start the timer running */
    PIT->CHANNEL[2].TCTRL |= PIT_TCTRL_TEN_MASK;
}

uint32_t counterStop(void) {
  uint32_t counter = 0;

  /* get the time elapsed since started */
  counter = 0XFFFFFFFF - PIT->CHANNEL[2].CVAL;

  /* Disable the timer */
  PIT->CHANNEL[2].TCTRL &= ~PIT_TCTRL_TEN_MASK;
  return counter;
}

void PIT2_IRQHandler(void) {
    /* Disable the timer */
    PIT->CHANNEL[2].TCTRL &= ~PIT_TCTRL_TEN_MASK;

    /* Disable interrupt on timeout */
    PIT->CHANNEL[2].TCTRL &= ~PIT_TCTRL_TIE_MASK;

    /* Clear the timer interrupt flag to allow further timer interrupts */
    PIT->CHANNEL[2].TFLG |= PIT_TFLG_TIF_MASK;

    /* We should never get here - timer overflow */
    assert(false);
}
```

Using an interval timer

```
int main () {
  uint32_t timeElapsed = 0;

  /* Initialise the counter. Only need to do this once */
  counterInit();

  /* Reset time elapsed */
  timeElapsed = 0;

  /* Start the counter */
  counterStart();

  /* Start of code to be measured */
  lcd.locate(2,8);
  lcd.printf("Hello world!\n");
  /* End of code to be measured */

  /* Stop the counter and retrieve time elapsed */
  timeElapsed = counterStop();

  /* Display the result */
  lcd.cls();
  lcd.locate(86, 8);
  lcd.printf("%u", timeElapsed);
}
```

Questions about the use of the interval timer

- How accurate is the timer?
  - Don't know. Don't have the equipment to calibrate it exactly. Read the data sheet for the clock crystal?
- How precise is the timer?
  - Don't know. Haven't done enough experiments yet. Need to perform multiple experiments. Compute mean and variance.
- What is the resolution of the timer?
  - Speed of peripheral clock for PIT1 is configured to be 0.5 of the SystemCoreClock, which is set at 120 MHz.
- What is the longest interval of time that can be measured with this configuration before the timer counter overflows (32 bit counter)?

Measuring execution time: summary

- There are numerous techniques for measuring software execution time.
- The use of an interval timer has been described here.
- Every approach to measurement has limitations and may introduce errors.
  - Whenever you present data based on measurement, it's important to discuss the likely accuracy, precision and resolution of the measurements.
  - Take care not to introduce systematic errors into your experiments, e.g. by perturbing the software that you are trying to measure.
  - Account for random errors by repetition of experiments to establish a level of confidence in the data.
- Worst-case execution time may not be revealed by measurement – results likely to be optimistic, ie to suggest a worst-case execution time that is less than the actual worst-case execution time.

Acknowledgements

- Raimund Kirner, Y.T. Li, S. Malik, Edward Lee, Sanjit Seshia, R. Wilhelm, *The Determination of Worst-Case Execution Times: Overview of Methods and Survey of Tools*, ACM Transactions on Embedded Computing Systems (TECS) 7:3, 2008