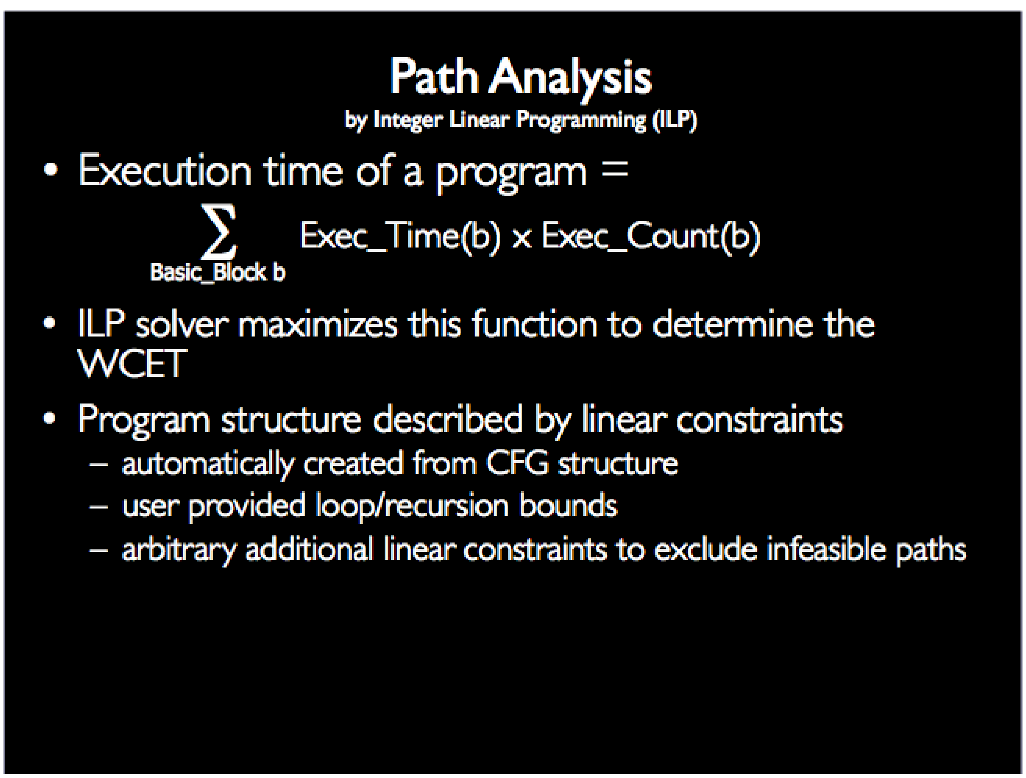
Rapita Tool Overview

Tomas Kalibera



Previous lecture (slides of Reinhard Wilhelm)

WCET Analysis with Rapita

1. Measurement
   - Instrumentation points inserted (i.e. down to basic block level)
   - Tests run on real hardware or a cycle accurate simulator
2. Structural analysis of source code
   - Identifies valid sequences of instrumentation points
3. Report generation
   - WCET estimation

In Rapita Reports...

- WCET
  - What is the WCET of function foo and which functions contribute to it the most / contribute at all
- Other Timings
  - High-water-mark execution time
    - Maximum execution time of a function when worst-case path of the root function was taken
  - Average execution time
  - Context sensitive
- Coverage
  - Loop bounds, call tree, context information

Call Tree

- Functions with blue bullets are on the worst-case path

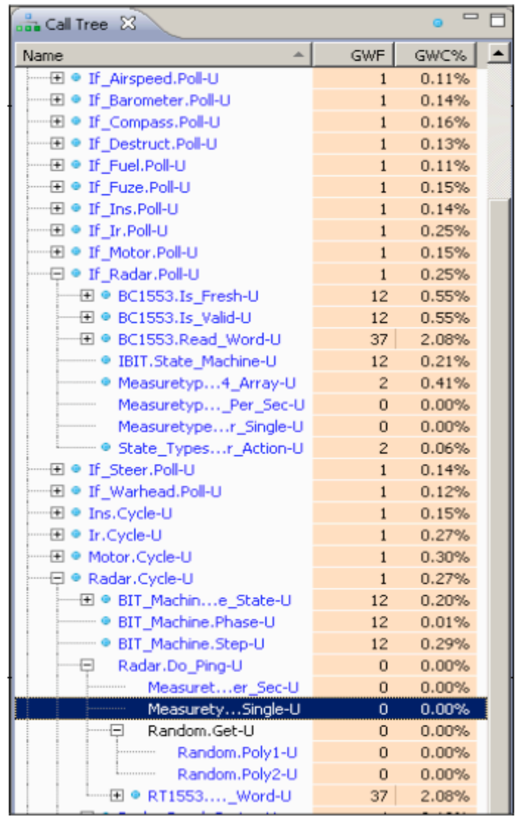


Figure from RapiTime White Paper

Source Code Display

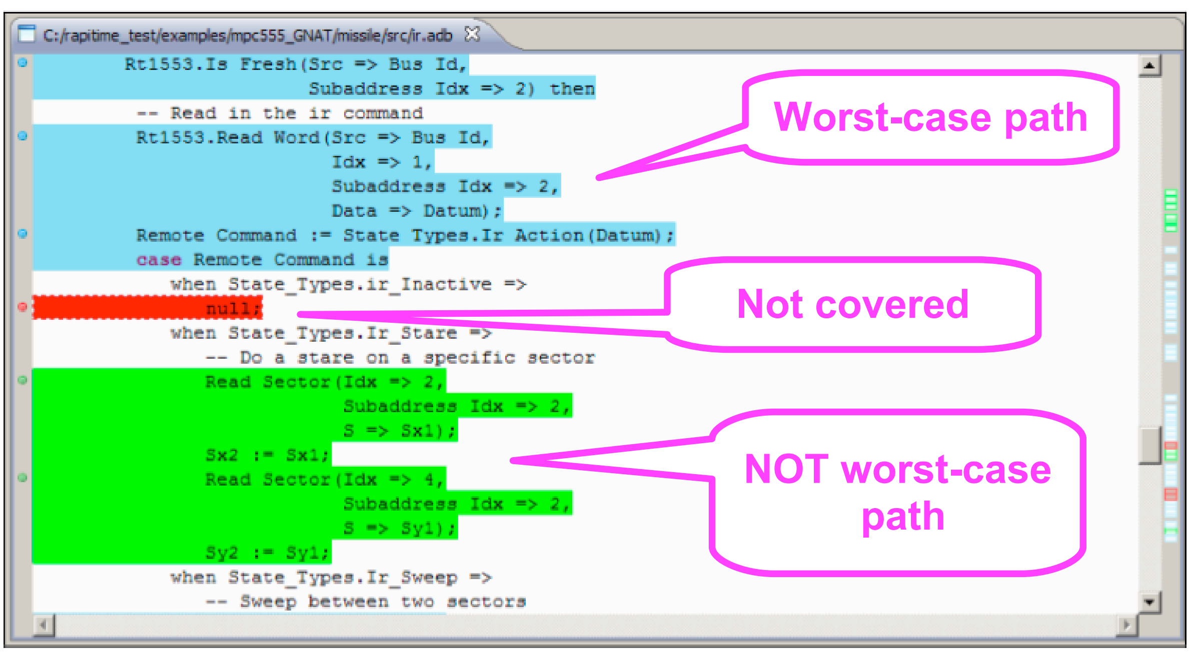


Figure from RapiTime White Paper

Execution Time Profile

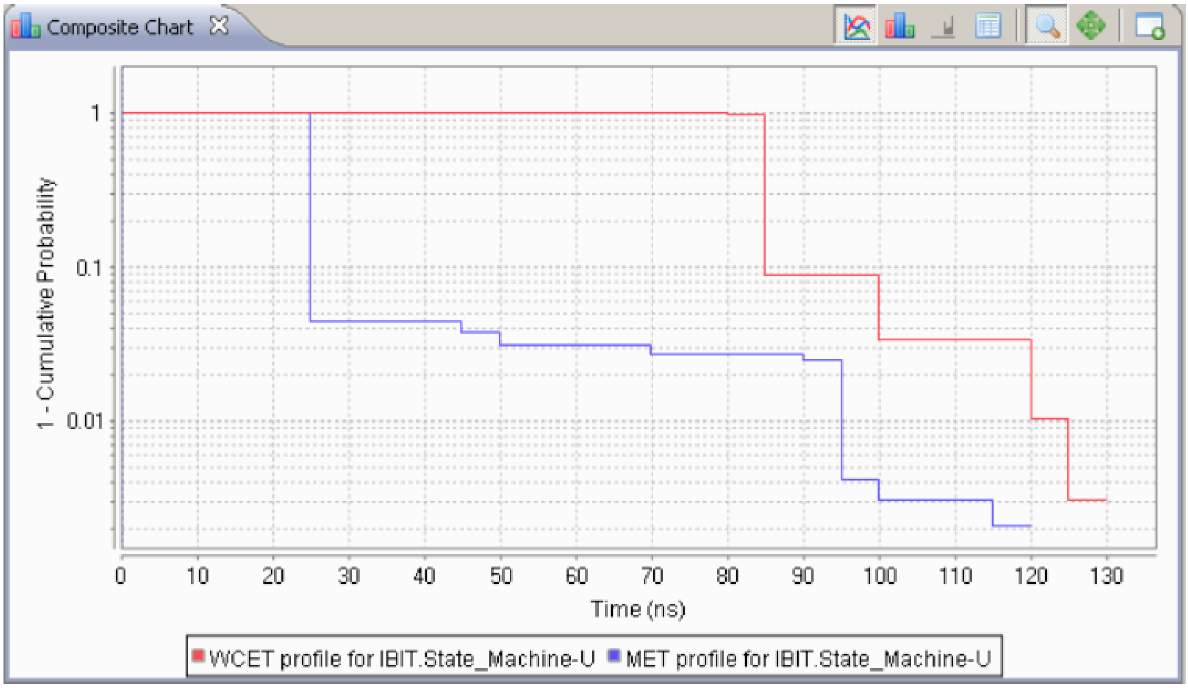


Figure from RapiTime White Paper

Code Instrumentation

- Instrumentation point
  - Unique integer id, individually time-stamped during measurement
- Insertion
  - Automatic via **cins** tool
    - Inserts used-defined code at each instrumentation point
    - The code is platform dependent and depends on tracing technique

Tracing Techniques

- On-target
  - Time-stamping and recording on target, buffering
  - Issues with buffer size, time-stamping overhead, overhead of dumping the results
  - Easy to implement
- Off-target
  - Time-stamping and recording off target
  - Requires a trace-box and low-latency harware interface on target

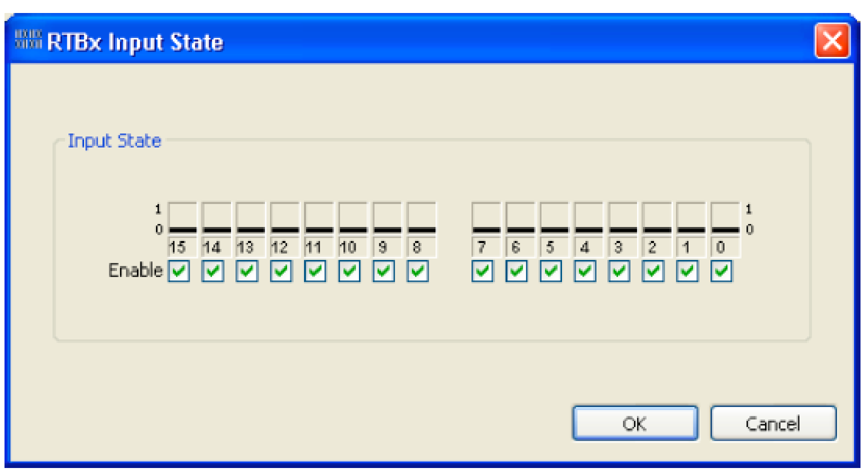
Tracing with Rapita Trace Box

Figure from RTBx documentation
Figure from GR-XC3S-1500 development board manual

Rapita Trace Box

- Interfaces
  - Flying leads connectors to the device
    - Grounded wire for every bit of input, external clock
  - Ethernet / TCPIP for control
    - Remote application for configuring trace jobs and debugging the device outputs
    - SAMBA share for downloading data
- Functionality
  - Reads, samples and stores traces at high frequency (ours works at 100Mhz, 16 bit of input data)

RTBx Control Center (checking the current inputs)



```
unsigned long base = 0x80000800;

uint32_t *output = (uint32_t *) (base+0x4);
uint32_t *outputMask = (uint32_t *) (base+0x8);
uint32_t *interruptMask = (uint32_t *) (base+0xc);

unsigned u;

printf("Initializing output port...\n");
*outputMask = 0xffff;
*interruptMask = 0;
*output = 0;

printf("Writing to output port...\n");

for(u=0;u<ITERATIONS;u++) {
  *output = u & 0xffff ;
}

*output = 0;
*outputMask = 0;
```

Testing RTBx with LEON3

Testing RTBx with LEON3

- RTBx config
  - Box internal clock, sampling at 80 MHz, oversampling by two (the LEON3 FPGA board runs at 40 MHz)
- Experiment steps
  - Configure and start trace job on RTBx
  - Start example app on the board
  - Stop the job, download compressed trace
  - Decompress the trace with traceutils
- Outcome
  - All of 50 mio loop iterations were correctly detected

Back to the WCET

(annotating code for better WCET estimates with Rapita)

Annotations: Loop Bounds

```
...
    for(j = 0; j<10; j++)
    {
#pragma RPT loop_max_iter (10);
      check (j);
    }
...
```

Figure from RapiTime User Guide

Local Worst-Case Frequency

```
...
    star_count = 0;
    for(i=0; i < limit; i++)
    {
#pragma RPT loop_max_iter (1000);
        if(buf[i] == '*')
        {
#pragma RPT wfreq (50);
            star_count++;
            if(star_count >= 50)
            {
#pragma RPT wfreq (1);
                save_count = star_count;
                break;
            }
        }
    }
...
```

Figure from RapiTime User Guide

Mutually Exclusive Paths

```
void my_fun( int a, int b )
{
#pragma RPT lwp_exactly_one_of ("path_A", "path_B");
    if(a == 0){
        #pragma RPT path_tag("path_A");
        ...
    }

    if (a !=0){
        #pragma RPT path_tag("path_B");
        ...
    }
}
```

Figure from RapiTime User Guide

Other Annotations

- Ignoring a path, mode specific paths
- Context dependencies (function arguments)
- Loop unrolling
- Black-box functions, specifying known WCET for a function
- Dealing with function pointers
- Dealing with recursion (direct requires bounds, indirect is not supported)