

Rapita Tool Overview

Tomas Kalibera

Path Analysis

by Integer Linear Programming (ILP)

- Execution time of a program =
$$\sum_{\text{Basic_Block } b} \text{Exec_Time}(b) \times \text{Exec_Count}(b)$$
- ILP solver maximizes this function to determine the WCET
- Program structure described by linear constraints
 - automatically created from CFG structure
 - user provided loop/recursion bounds
 - arbitrary additional linear constraints to exclude infeasible paths

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

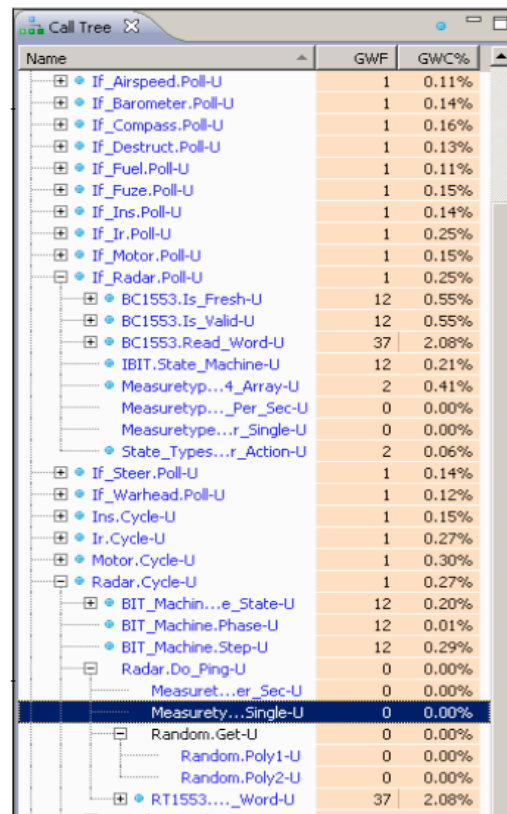
- WCET estimation

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



Name	GWF	GWC%
If_Airspeed.Poll-U	1	0.11%
If_Barometer.Poll-U	1	0.14%
If_Compass.Poll-U	1	0.16%
If_Destruct.Poll-U	1	0.13%
If_Fuel.Poll-U	1	0.11%
If_Fuze.Poll-U	1	0.15%
If_Ins.Poll-U	1	0.14%
If_Ir.Poll-U	1	0.25%
If_Motor.Poll-U	1	0.15%
If_Radar.Poll-U	1	0.25%
BC1553.Is_Fresh-U	12	0.55%
BC1553.Is_Valid-U	12	0.55%
BC1553.Read_Word-U	37	2.08%
IBIT.State_Machine-U	12	0.21%
Measuretyp...4_Array-U	2	0.41%
Measuretyp...Per_Sec-U	0	0.00%
Measuretype...r_Single-U	0	0.00%
State_Types...r_Action-U	2	0.06%
If_Steer.Poll-U	1	0.14%
If_Warhead.Poll-U	1	0.12%
Ins.Cycle-U	1	0.15%
Ir.Cycle-U	1	0.27%
Motor.Cycle-U	1	0.30%
Radar.Cycle-U	1	0.27%
BIT_Machin...e_State-U	12	0.20%
BIT_Machine.Phase-U	12	0.01%
BIT_Machine.Step-U	12	0.29%
Radar.Do_Ping-U	0	0.00%
Measuret...er_Sec-U	0	0.00%
Measurety...Single-U	0	0.00%
Random.Get-U	0	0.00%
Random.Poly1-U	0	0.00%
Random.Poly2-U	0	0.00%
RT1553...._Word-U	37	2.08%

Figure from RapiTime White Paper

Source Code Display

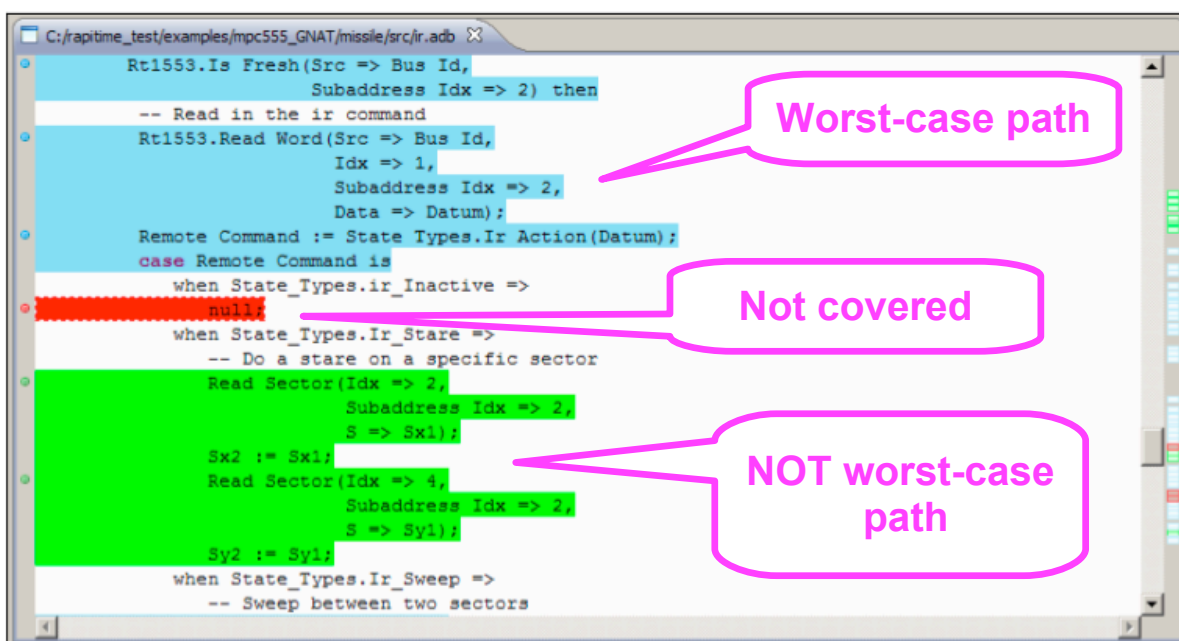
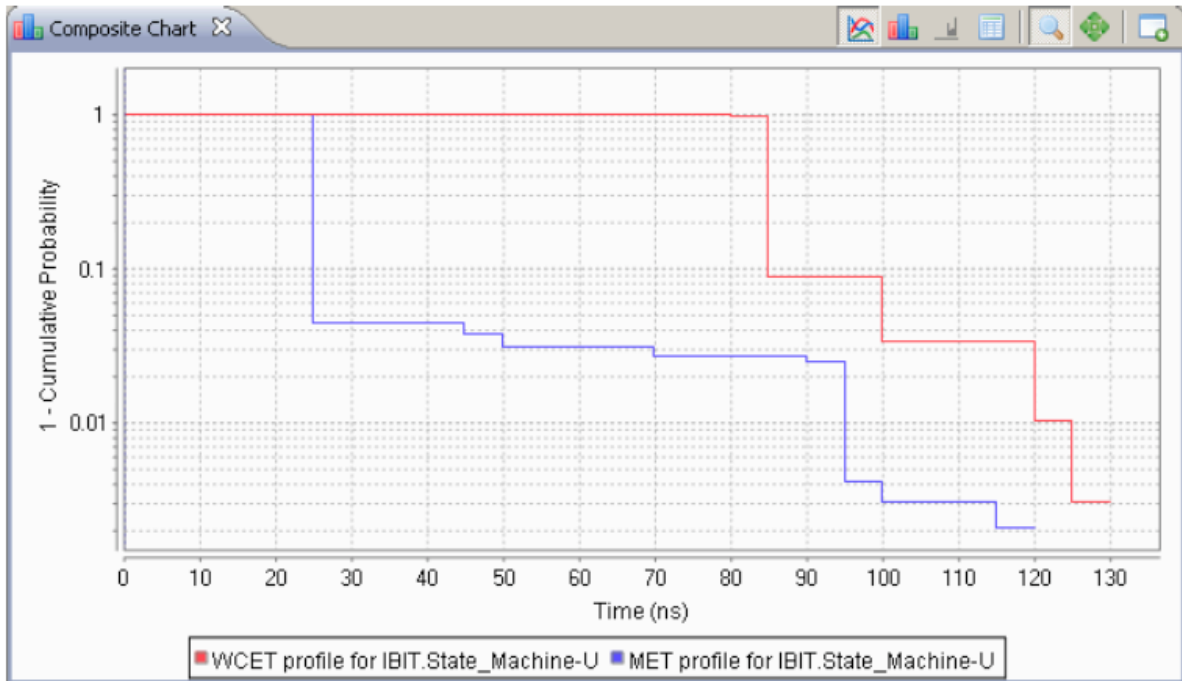


Figure from RapiTime White Paper

Execution Time Profile



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 hardware interface on target

Tracing with Rapita Trace Box



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

Rapita Trace Box

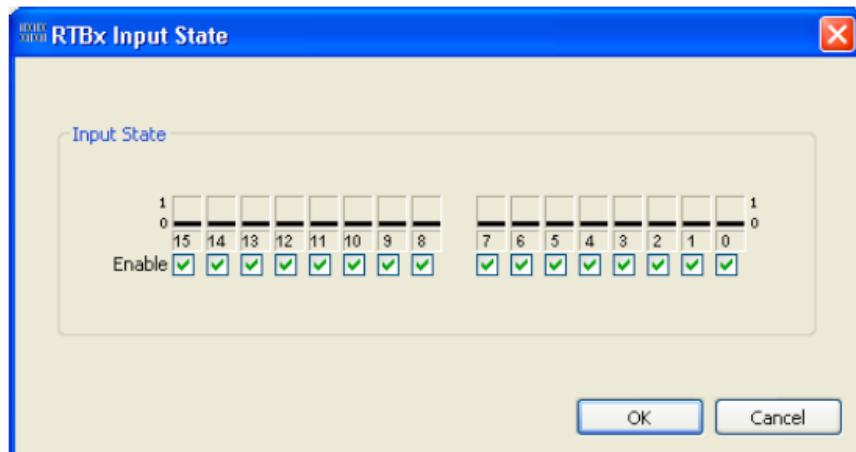
- Interfaces

- 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);  
    }  
...
```


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)