Outline

1. Introduction and Motivation
2. Contribution
3. WCET Analysis by SMT Encoding
   - Naive SMT Approach
   - Offset-based SMT Encoding
4. Performance Evaluation
5. Conclusion and Future Work
Introduction: Static Timing Analysis

- Used mainly in safety critical systems
- Requires strong guarantees on timing constraints (hard real-time)

Static Timing Analysis

- Guarantee timing constraints and deadlines
- **Challenge:** Reduce the over-approximation
Shared resources: contention, access delays

Time Division Multiple Access (TDMA) arbitration policy
  - Time triggered policy
  - Assign fixed time slots to each core
Introduction: Time Division Multiple Access

Core A viewpoint:

- **Time**
- **Core A**
- **Core B**
- **Core C**

- **Core A**
- **Core B**

- **T**
- **acc**
- **execution time of a granted request**
- **W**
- **waiting time to grant requests**
  - \( \in [0, \pi - (\sigma - \text{acc})] \)

- **Pessimistic approach**

- **Offset**
- **req**

- **W**
- **Offset relative to the TDMA period**

- **req**

- **Time**
Introduction: Time Division Multiple Access

Core A viewpoint:

- **Time**:
  - Core A
  - Core B
  - Core C
  - Core A
  - Core B
- **Slot length**: $\sigma$
- **Period**: $\pi$

**Execution time** of a granted request $\cdot$ **Waiting time** to grant requests $\in [0, \pi - (\sigma - \text{acc})]$.

**Pessimistic approach**: Worst-Case$(T) = \pi - (\sigma - \text{acc})$.

**Offset relative to the TDMA period**: $\text{Offset}(\text{req}) = \text{time}_{\text{instant}}(\text{req}) \mod \pi$. 
Introduction: Time Division Multiple Access

Core A viewpoint:

- Slot length \( \sigma \)
- Period \( \pi \)
- Request \( \text{req}_1 \)
- Response \( \text{resp} \)
- Acceptance \( \text{acc} \)

Execution time of a granted request: \( T \)
Waiting time to grant requests: \( \in [0, \pi - (\sigma - \text{acc})] \)

Pessimistic approach:

\[
\text{Worst-Case}(T) = \pi - (\sigma - \text{acc})
\]
**Introduction: Time Division Multiple Access**

Core A viewpoint:

![Diagram showing Core A viewpoint with time division multiple access (TDMA) slots and periods.](image)

- **req₁**: Request 1
- **resp**: Response
- **acc**: Accept
- **T**: Total execution time
- **σ**: Slot length
- **π**: Period

**Execution Time Calculation**:

- **T_{Grant}**: Execution time of a granted request
- **T_{Waiting}**: Waiting time to grant requests

**Pessimistic Approach**:

- **Worst-Case (T)**: \( \pi - (\sigma - \text{acc}) \)

**Model Details**:

- Offset relative to the TDMA period:
  \( \text{Offset} \text{(req)} = \text{time} \text{instant} \text{(req)} \mod \pi \)
Introduction: Time Division Multiple Access

Core A viewpoint:

- \(acc\) execution time of a granted request
- \(T\) waiting time to grant requests \(\in [0, \pi - (\sigma - acc)]\)

**Pessimistic approach:** \(\text{Worst-Case}(T) = \pi - (\sigma - acc)\)
Introduction: Time Division Multiple Access

Core A viewpoint:

- $acc$ execution time of a granted request
- $T$ waiting time to grant requests $\in [0, \pi - (\sigma - acc)]$

**Pessimistic approach:** $Worst-Case(T) = \pi - (\sigma - acc)$

Offset relative to the TDMA period:

$$Offset(req) = time_{\text{instant}}(req) \mod \pi$$
Motivation: WCET Analysis of TDMA

Goal: Estimate the WCET

```
/* 3 cycles */
if (cond)

BB 2
/* 10 cycles*/

if (¬cond)

/* bus access */

BB 5
/* 5 cycles */
return()
```

Degree:

- Option 1: Worst-case everywhere.
- Option 2: Capture all possible offsets
- Option 3: Feasible Path Analysis
Motivation: WCET Analysis of TDMA

Goal: Estimate the WCET

- Option 1: Worst-case everywhere.

$\text{cost} = \pi - \sigma + 2 \text{acc}$
Motivation: WCET Analysis of TDMA

Goal: Estimate the WCET
- Option 1: Worst-case everywhere.
- Option 2: Capture all possible offsets
Motivation: WCET Analysis of TDMA

Goal: Estimate the WCET
- Option 1: Worst-case everywhere.
- Option 2: Capture all possible offsets
- Option 3: Feasible Path Analysis
Contribution:

Compute WCET by encoding the semantics and shared resource accesses into an optimization problem (SMT)

TDMA Bus Analysis
Kelter et al., RTS’14
Chattopadhyay et al., SCOPES’10

Feasible Path Analysis with SMT
Henry et al. LCTES’14
Contribution:

Compute WCET by encoding the semantics and shared resource accesses into an optimization problem (SMT)
WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem

- SMT query
  - \( \text{assert} \left( \bigwedge \text{expr} \right) = \text{"Is there a trace with correct semantics?"} \) such that the execution time is greater than \( X \)

- SMT-solver response:
  - SAT
  - UNSAT

Goal: Find the smallest \( X \) such that Execution Time \( > X \) is UNSAT

8
WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem

  \[
  \text{SMT query} = \text{"Is there a trace with correct semantics ?"} \\
  \text{assert}(\land \text{expr})
  \]

- SMT-solver response:
  - SAT: There is a feasible execution path
  - UNSAT: There is no feasible execution path

Goal: Find the smallest \( X \), such that \( \text{Execution Time} > X \) is UNSAT
WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem
- Add execution times on the paths
  - SMT query: “Is there a trace with correct semantics?”
    \[ \text{assert}(\bigwedge \text{expr}) \]
- SMT-solver response:
  - SAT:
  - UNSAT:

Goal: Find the smallest \( X \), such that \( \text{Execution Time} > X \) is UNSAT.
WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem
- Add execution times on the paths

\[
\text{assert(}\land \text{expr)}
\]

...such that the execution time is greater than \( X \)

- SMT-solver response:
  - SAT: There is a feasible path with an execution time > \( X \)
  - UNSAT: We found a WCET upper bound
WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem
- Add execution times on the paths
  
  \[
  \text{SMT query} = \text{“Is there a trace with correct semantics}\]
  
  \[
  \text{assert}(\land \text{expr})
  \]

  ...such that the execution time is greater than $X$?

- SMT-solver response:
  - SAT: There is a feasible path with an execution time $> X$
  - UNSAT: We found a WCET upper bound

Goal:

Find the smallest $X$, such that Execution Time $> X$ is UNSAT
Example: Semantics and Timing Encoding

```plaintext
B_1
y = read_value()
if (y < 0)

pred = (y < 0)
t_{1,2} = b_1 \land \text{pred}

B_2
/* 10 cycles*/

if (y \geq 0)

b_2 = t_{1,2}

B_3

b_3 = t_{1,3} \lor t_{2,3}

B_4
/* bus access */

b_5

b_i "true" if B_i executed

\text{t}_{i,j} "true" if transition B_i \rightarrow B_j taken
```
Example: Semantics and Timing Encoding

\[ y = \text{read\_value()} \]

if \((y < 0)\)

\[
\begin{align*}
\text{pred} &= (y < 0) \\
t_{1,2} &= b_1 \land \text{pred} \\
e_{1,2} &= \text{start} + \text{wcet}(B_1)
\end{align*}
\]

\[ b_2 = t_{1,2} \]

\[ /\ast 10 \text{ cycles}\ast / \]

\[ e_{2,3} = e_{1,2} + 10 \]

if \((y \geq 0)\)

\[ b_3 = t_{1,3} \lor t_{2,3} \]

\[ t_{3,5} = \text{start} + \text{wcet}(B_3) \]

\[ e_{3,5} = e_{3,4} + 10 \]

\[ b_4 = t_{3,5} \lor t_{2,3} \]

\[ /\ast \text{bus access}\ast / \]

\[ b_5 = t_{4,5} \lor t_{2,3} \]

\[ \text{execution time} = \begin{cases} 
\text{if } t_{3,5} \text{ then } e_{3,5} \text{ else } e_{4,5} 
\end{cases} \]

\[ \begin{align*}
\text{start} &= 0 \\
\text{pred} &= (y < 0) \\
t_{1,2} &= b_1 \land \text{pred} \\
e_{1,2} &= \text{start} + \text{wcet}(B_1)
\end{align*} \]

- \( b_i \) "true" if \( B_i \) executed
- \( t_{i,j} \) "true" if transition \( B_i \rightarrow B_j \) taken
- \( e_{i,j} \) execution time at transition \( B_i \rightarrow B_j \)
Example: Semantics and Timing Encoding

\[
\begin{align*}
\text{start} &= 0 \\
\text{B}_1 &\quad y = \text{read\_value()} \\
&\quad \text{if } (y < 0) \\
&\quad e_{1,2} = \text{start} + \text{wcet}(\text{B}_1) \\
\text{B}_2 &\quad /* 10 cycles*/ \\
&\quad e_{2,3} = e_{1,2} + 10 \\
\text{B}_3 &\quad \text{if } (y \geq 0) \\
\text{B}_4 &\quad /* \text{bus access} */ \\
\text{B}_5 &
\end{align*}
\]

\[e_{i,j} \text{ execution time at transition } \text{B}_i \rightarrow \text{B}_j\]

\[
\text{execution time} = \text{if } t_{3,5} \text{ then } e_{3,5} \text{ else } e_{4,5}
\]
Example: Semantics and Timing Encoding

\[
\text{start} = 0
\]

\[
y = \text{read\_value}()
\]

\[
\text{if } (y < 0)
\]

\[
e_{1,2} = \text{start} + \text{wcet}(B_1)
\]

\[
B_2
\]

\[
/* 10 cycles*/
\]

\[
e_{2,3} = e_{1,2} + 10
\]

\[
B_3
\]

\[
\text{if } (y \geq 0)
\]

\[
B_4
\]

\[
/* \text{bus access} */
\]

\[
e_{4,5} = e_{3,4} + \text{tdma\_cost}(e_{3,4})
\]

\[
B_5
\]

\[
\text{execution time} = \text{if } t_{3,5} \text{ then } e_{3,5} \text{ else } e_{4,5}
\]

\textbf{Example: Semantics and Timing Encoding}

\[
e_{i,j} \text{ execution time at transition } B_i \rightarrow B_j
\]

\[
\text{tdma\_cost()} \text{ execution time of a bus access}
\]
Naive SMT Encoding

Algorithm \textit{tdma\_cost}: returns the execution time of a bus access

Require: \( e_{\text{entry}} \)

1: \( \text{offset}_{\text{entry}} \leftarrow e_{\text{entry}} \mod \pi \)
2: if \( \text{offset}_{\text{entry}} \in [0, \sigma - \text{acc}] \) then /* offset is inside the slot */
3: \( \text{return acc} \)
4: else /* offset is outside the slot */
5: \( \text{return } (\pi - \text{offset}_{\text{entry}}) + \text{acc} \)
6: end if
Performance of the Naive Encoding
Performance of the Naive Encoding

The figure shows the time (in seconds on a log scale) required to perform $N$ blocks of access for $N = 1, 10, 100, 1000$. The line represents $100\%$ access (naive). As $N$ increases, the time also increases significantly, indicating a high cost associated with naive encoding.
Naive SMT Encoding

Algorithm $tdma\_cost$: returns the execution time of a bus access

Require: $e_{entry}$

1: $offset_{entry} \leftarrow e_{entry} \mod \pi$
2: if $offset_{entry} \in [0, \sigma - acc]$ then /* offset is inside the slot */ ①
3: return $acc$
4: else /* offset is outside the slot */ ②
5: return $(\pi - offset_{entry}) + acc$
6: end if
Offset-based SMT Encoding

\[
\begin{align*}
\text{start} & = 0 \\
B_1 & \\
& y = \text{read\_value}() \\
& \text{if} \ (y < 0) \\
& e_{1,2} = \text{start} + \text{wcet}(B_1) \\
& \text{if} \ (y \geq 0) \\
B_3 & \\
& /* \text{bus access} */ \\
& e_{2,3} = e_{1,2} + 10 \\
& B_4 \\
& e_{4,5} = e_{3,4} + \text{tdma\_cost}(\text{off}_{3,4}) \\
B_5 & \\
\text{execution time} & = \text{if} \ t_{3,5} \ \text{then} \ e_{3,5} \ \text{else} \ e_{4,5}
\end{align*}
\]

\[\text{off}_{i,j} = e_{i,j} \mod \pi\]
Offset-based SMT Encoding

\[
\text{start} = 0 \\
\text{off}_s \in [0, \pi[ \\
B_1 \\
y = \text{read\_value()} \\
\text{if } (y < 0) \\
e_{1,2} = \text{start} + \text{wcet}(B_1) \\
\text{off}_{1,2} = (\text{off}_s + \text{wcet}(B_1)) \mod \pi \\
B_2 \\
/* 10 cycles */ \\
e_{2,3} = e_{1,2} + 10 \\
\text{off}_{2,3} = (\text{off}_{1,2} + 10) \mod \pi \\
B_3 \\
\text{if } (y \geq 0) \\
B_4 \\
/* \text{bus access} */ \\
e_{4,5} = e_{3,4} + \text{tdma\_cost}(\text{off}_{3,4}) \\
\text{off}_{4,5} = \text{tdma\_offset}(\text{off}_{3,4}) \\
B_5 \\
\text{execution time} = \text{if } t_{3,5} \text{ then } e_{3,5} \text{ else } e_{4,5} \\
\]

- \( \text{off}_{i,j} = e_{i,j} \mod \pi \)
- \( \text{off}_{i,j} \) offset at transition \( B_i \rightarrow B_j \)
Offset-based SMT Encoding

Algorithm $tdma\_cost$: returns the execution time of a bus access

Require: $\epsilon_{entry}$ offset$_{entry}$

1: $\text{offset}_{entry} \leftarrow \epsilon_{entry} \mod \pi$
2: $\text{if } \text{offset}_{entry} \in [0, \sigma - \text{acc}] \text{ then }/* \text{ offset is inside the slot }*/$
3: $\text{return acc}$
4: $\text{else }/* \text{ offset is outside the slot }*/$
5: $\text{return } (\pi - \text{offset}_{entry}) + \text{acc}$
6: $\text{end if}$
Algorithm *tdma_cost*: returns the execution time of a bus access

Require: `offset_{entry}`

1: \[\text{if } \text{offset}_{\text{entry}} \in [0, \sigma - \text{acc}] \text{ then } /* \text{offset is inside the slot} */\]
2: \[\text{return } \text{acc} \]
3: \[\text{else } /* \text{offset is outside the slot} */\]
4: \[\text{return } (\pi - \text{offset}_{\text{entry}}) + \text{acc} \]
5: \[\text{end if} \]
Offset-based SMT Encoding

Algorithm \textit{tdma\_offset}: returns the offset after a bus access

Require: $\text{offset}_{\text{entry}}$

1: if $\text{offset}_{\text{entry}} \in [0, \sigma - \text{acc}]$ then /* offset is inside the slot */
2: \hspace{1em} return $\text{offset}_{\text{entry}} + \text{acc}$
3: else /* offset is outside the slot */
4: \hspace{1em} return $\text{acc}$
5: end if
y = read_value()
if (y < 0)

/* 10 cycles*/

if (y ≥ 0)

/* bus access */

off_{2,3} = (off_{1,2} + 10) \mod \pi
Offset-based SMT Encoding

\[
y = \text{read\_value()}
\]

\[
\text{if (y < 0)}
\]

\[
B_1
\]

\[
/* 10 cycles*/
\]

\[
B_2
\]

\[
\text{if (y} \geq 0)
\]

\[
B_3
\]

\[
/* \text{bus access} */
\]

\[
B_4
\]

\[
B_5
\]

\[
\text{off}_{2,3} = (\text{off}_{1,2} + 10) \mod \pi
\]

\[
\text{execution time} = \begin{cases} 
\text{if } t_{3,5} \text{ then } e_{3,5} \text{ else } e_{4,5} \\
\end{cases}
\]

\[
<_{\pi} (\text{off}_{i,j} + c) \mod \pi
\]

\[
\text{def } \alpha < 2\pi
\]

\[
(\text{off}_{i,j} + c \mod \pi) \mod \pi
\]

\[
\text{if } \alpha < \pi \text{ then } \alpha \text{ else } \alpha - \pi
\]
Offset-based SMT Encoding

\[
\text{start} = 0 \\
\text{off}_s \in [0, \pi]
\]

\[
y = \text{read\_value()}
\]

if \((y < 0)\)

\[
B_1
\]

\[
e_{1,2} = \text{start} + \text{wcet}(B_1)
\]

/* 10 cycles*/

\[
e_{2,3} = e_{1,2} + 10 \\
\text{off}_{2,3} = \text{get\_offset}(\text{off}_{1,2}, 10)
\]

if \((y \geq 0)\)

\[
B_3
\]

\[
\text{off}_{3,5}
\]

/* bus access */

\[
B_4
\]

\[
e_{4,5} = e_{3,4} + \text{tdma\_cost}(\text{off}_{3,4}) \\
\text{off}_{4,5} = \text{tdma\_offset}(\text{off}_{3,4})
\]

\[
B_5
\]

execution time = if \(t_{3,5}\) then \(e_{3,5}\) else \(e_{4,5}\)
Performance of the Offset-based Encoding

\[ N \]

- Access
- Access
- Access

Graph:
- 100% access (naive)
- 100% access
- 0% access

- Time (s) (log scale)
- # basic blocks (log scale)
Proof-of-Concept Implementation

C code → LLVM compiler → LLVM bitcode

Timing model → SMT clauses → PAGAI

→ WCET

SMT-solving
Proof-of-Concept Implementation

C code → LLVM compiler → LLVM bitcode:
- Unrolled loops

Timing model:
- 1 instruction = 1 cycle
- Each load and store requests a bus access
- Timing Composable

LLVM compiler → SMT clauses → PAGAI

SMT-solving → WCET
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>#LLVM instr.</th>
<th>#bus access</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>Binary search</td>
<td>231</td>
<td>12</td>
</tr>
<tr>
<td>insertsort</td>
<td>Insertion sort on a reversed array</td>
<td>493</td>
<td>65</td>
</tr>
<tr>
<td>jfdctint</td>
<td>Discrete Cosine Transformation</td>
<td>2334</td>
<td>448</td>
</tr>
<tr>
<td>fdct</td>
<td>Fast Discrete Cosine Transform</td>
<td>2502</td>
<td>385</td>
</tr>
<tr>
<td>compressdata</td>
<td>Data compression program adopted from SPEC95</td>
<td>674</td>
<td>131</td>
</tr>
<tr>
<td>fly-by-wire</td>
<td>UAV fly-by-wire software</td>
<td>2815</td>
<td>515</td>
</tr>
</tbody>
</table>

Table: Benchmark description
Estimated WCET of the benchmarks with different configurations of the TDMA bus.
### Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>(&lt;40, 20, 10)&gt;</th>
<th>(&lt;400, 100, 40)&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bs</td>
<td>0.45s</td>
<td>0.98s</td>
</tr>
<tr>
<td>insertsort</td>
<td>1.37s</td>
<td>6.56s</td>
</tr>
<tr>
<td>jfdctint</td>
<td>44.10s</td>
<td>48.54s</td>
</tr>
<tr>
<td>fdct</td>
<td>41.36s</td>
<td>34.57s</td>
</tr>
<tr>
<td>compressdata</td>
<td>4.66s</td>
<td>3.23s</td>
</tr>
<tr>
<td>fly-by-wire</td>
<td>28.78s</td>
<td><strong>149.01s</strong></td>
</tr>
</tbody>
</table>

Analysis time
Future Work

C code:
- Compositional analysis
- Partial loop unrolling

Executable binary

LLVM compiler

Timing model:
- Realistic timing values (e.g. Ottawa)
- Cache analysis

SMT clauses

SMT-solving

WCET

PAGAI

LLVM bitcode

Executable binary

LLVM compiler

Timing model:
- Realistic timing values (e.g. Ottawa)
- Cache analysis

SMT clauses

SMT-solving

WCET

PAGAI

LLVM bitcode
Summary

- SMT encodings for TDMA access
- Comparison between different encodings
- Validation with small but relevant benchmarks

Why does it work?

- Feasible Path Analysis combined with the WCET computation
- Best-case of gain: All requests are within the TDMA slots
- Worst-case of gain: All requests have worst-case delay
Summary

- SMT encodings for TDMA access
- Comparison between different encodings
- Validation with small but relevant benchmarks

Why does it work?

- Feasible Path Analysis combined with the WCET computation
- Best-case of gain: All requests are within the TDMA slots
- Worst-case of gain: All requests have worst-case delay

⇝ SMT is an interesting research direction for WCET Analysis
WCET Analysis in Shared Resources Real-Time Systems with TDMA Buses

Hamza Rihani¹, Matthieu Moy¹, Claire Maiza¹ and Sebastian Altmeyer²

¹Univ. Grenoble Alpes / Verimag
²University of Luxembourg

J. Henry, M. Asavoae, D. Monniaux, and C. Maïza. How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. LCTES ’14, NY, USA.

1 instruction = 1 cycle

\[
< \pi, \sigma, acc > = < 6, 2, 1 >
\]
New Encoding

\[
\text{start} = 0 \\
\text{off} \_ s \in [0, \pi[ \\
\text{B}_1 \\
y = \text{read\_value()} \\
\text{if} (y < 0) \\
\text{B}_2 \\
\text{/* 10 cycles*/} \\
c_1,2 = \text{ite} \ t_1,2 \ wcet(B_1) \ 0 \\
\text{B}_3 \\
\text{if} (y \geq 0) \\
\text{B}_4 \\
\text{/* bus access */} \\
c_2,3 = \text{ite} \ t_2,3 \ 10 \ 0 \\
\text{off}_{2,3} = \text{get\_offset}(\text{off}_{1,2,10}) \\
c_4,5 = \text{ite} \ t_4,5 \ \text{tdma\_cost(} \text{off}_{3,4} \text{) 0} \\
\text{off}_{4,5} = \text{tdma\_offset(} \text{off}_{3,4} \text{)} \\
\text{B}_5 \\
\text{execution time} = \sum c_{i,j}
\]

Encode the costs of the basic blocks
\[e_{i,j} \text{ (absolute time)} \rightarrow c_{i,j} \text{ (cost)}\]

"ite C A B" ⇔ “if C then A else B"
Using TDMA functions

- **if..then..else encoding**
  
  \[ \text{off} = \text{ite} \ t_{13} \ \text{off}_{13} \ \text{off}_{23} \]
  
  \[ \text{off}_{35} = \text{off} \]
  
  \[ \text{off}_{34} = \text{off} \]

- **sum encoding**
  
  \[ \text{off} = \text{off}_{13} + \text{off}_{23} \]
  
  \[ \text{off}_{35} = \text{ite} \ t_{35} \ \text{off} \ 0 \]
  
  \[ \text{off}_{34} = \text{ite} \ t_{34} \ \text{off} \ 0 \]
Using TDMA functions

if..then..else (ite) encoding:

\[
\text{off}_{i,j} = (\text{if } t_{1,i} \text{ then } \text{off}_{1,i} \\
\text{else if } t_{2,i} \text{ then } \text{off}_{2,i} \\
\text{else} \ldots \\
\text{else if } t_{K,i} \text{ then } \text{off}_{K,i} \text{ else } 0)
\]

sum encoding:

\[
\text{off}_i = \sum_{k=1}^{K} \text{off}_{k,i} \\
\text{off}_{i,j} = \text{if } t_{i,j} \text{ then } \text{off}_i \text{ else } 0
\]
Performance 3

\[ \text{block} \times N \]

- Graph showing time (s) (log scale) vs. \( N \) for the 'sum' and 'ite' operations.
How it works?

- Example with binary search:

Testing $\text{wcet} \geq 0$... SAT (value found = 18).

New interval = $[18, 73]$.

Testing $\text{wcet} \geq 46$... UNSAT. New interval = $[18, 45]$.

Testing $\text{wcet} \geq 32$... UNSAT. New interval = $[18, 31]$.

Testing $\text{wcet} \geq 25$... UNSAT. New interval = $[18, 24]$.

Testing $\text{wcet} \geq 21$... UNSAT. New interval = $[18, 20]$.

Testing $\text{wcet} \geq 19$... UNSAT. New interval = $[18, 18]$.

The maximum value of $\text{wcet}$ is 18.

Computation time is 0.010000s