Compiling avionics software with a formally verified compiler

Sandrine Blazy

IFIP WG 2.11, 2014-03-18

The CompCert project
compcert.inria.fr

Goal: develop and prove correct a realistic compiler usable for critical embedded software

- from (a very large subset of) the C language
- to assembly code for popular processors (PowerPC, ARM, x86)
- producing reasonable efficient code (→ some optimizations)

Used Coq to mechanize the proof of semantic preservation and also to implement most of the compiler.
(Executable via automatic extraction to Caml.)

Verifying a compiler

Using Coq, we prove the following semantic preservation property:

> For all source programs S and compiler-generated code C,
> if the compiler generates machine code C from source S,
> without reporting a compilation error,
> then «C behaves like S».

- Compilers are allowed to fail (ill-formed source code, or capacity exceeded).

- Compiler written from scratch, along with its proof; not trying to prove an existing compiler.

Verifying a compiler

Using Coq, we prove the following semantic preservation property:

> For all source programs S and compiler-generated code C,
> if the compiler generates machine code C from source S,
> without reporting a compilation error,
> then «C behaves like S».

- Compilers are allowed to fail (ill-formed source code, or capacity exceeded).
- Compiler written from scratch, along with its proof; not trying to prove an existing compiler.

Reducing non-determinism during compilation

Languages such as C leave evaluation order partially unspecified.

```
int x = 0;
int f(void) { x = x + 1; return x; }
int g(void) { x = x - 1; return x; }
```

The expression `f()+g()` can evaluate either to:

- 1 if `f()` is evaluated first (returning 1), then `g()` (returning 0);
- -1 if `g()` is evaluated first (returning -1), then `f()` (returning 0).

Every C compiler chooses one evaluation order at compile-time.
The compiled code therefore has fewer behaviors than the source program (1 instead of 2).

Semantics preservation property

For all source programs S and compiler-generated code C,
if the compiler generates machine code C from source S,
without reporting a compilation error,
then the observable behavior of C is one of the possible observable behaviors of S according to the C semantics.

Semantics preservation property

For all source programs S and compiler-generated code C,
if the compiler generates machine code C from source S,
without reporting a compilation error,
then the observable behavior of C is one of the possible observable behaviors of S according to the C semantics.

Behaviors = termination / divergence / undefined («going wrong»)
+ trace of I/O operations performed

Semantics preservation property

For all source programs S and compiler-generated code C,
if the compiler generates machine code C from source S,
without reporting a compilation error,
then the observable behavior of C is one of the possible observable behaviors of S according to the C semantics, or improves on one of these possible behaviors.

Behaviors = termination / divergence / undefined («going wrong»)
+ trace of I/O operations performed

Improving = replacing undefined behaviors by more defined behavior

Improving behaviors during compilation

```
#include <stdio.h>
int main()
{
    int x;
    printf("Crash!\n");
    x = 1 / 0;
    return 0;
}
```

Compilers routinely optimize away going-wrong behaviors.

This program goes wrong.

However, the compiler eliminates `x=1/0;` as it is dead code.

Thus, the generated code terminates with the same trace of observable events `out("Crash!\n")`.

Improving behaviors during compilation

```
#include <stdio.h>
int main()
{
   int x[2] = { 12, 34 };
   printf("x[2] = %d\n", x[2]);
   return 0;
}
```

This program goes wrong.

However, the code generated by the compiler does not check the array bounds.

The generated code may crash but in general it prints an arbitrary integer and terminates normally.

A consequence of the main CompCert theorem

We know that the source program does not go wrong

- e.g. because it was formally verified with a static analyzer.

If the source program can not go wrong, then the behavior of the generated assembly code is exactly one of the behaviors of the source program.

```
Theorem transf_c_program_is_refinement:
forall p tp, transf_c_program p = OK tp →
(forall behv, exec_C_program p behv → not_wrong behv) →
(forall behv,  exec_Asm_program tp behv → exec_C_program p behv).
```

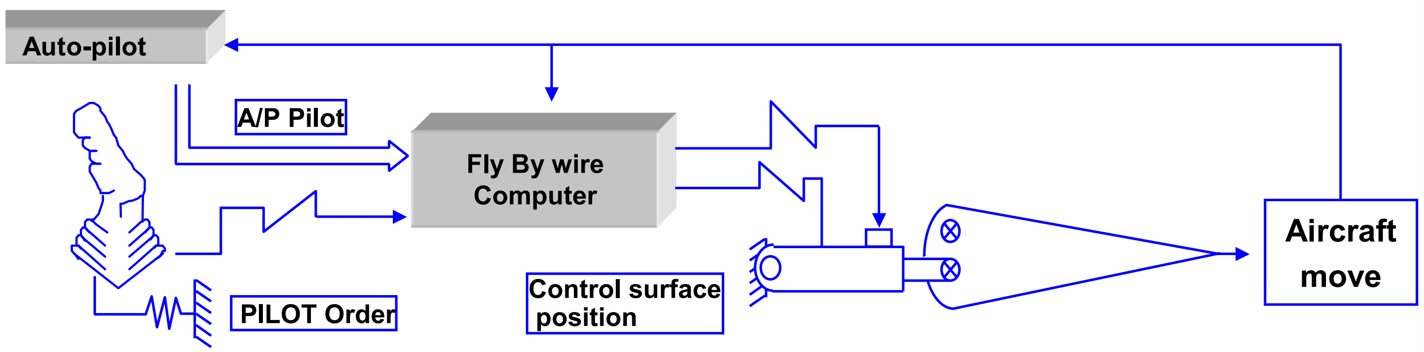
The generated assembly code can not wrong.

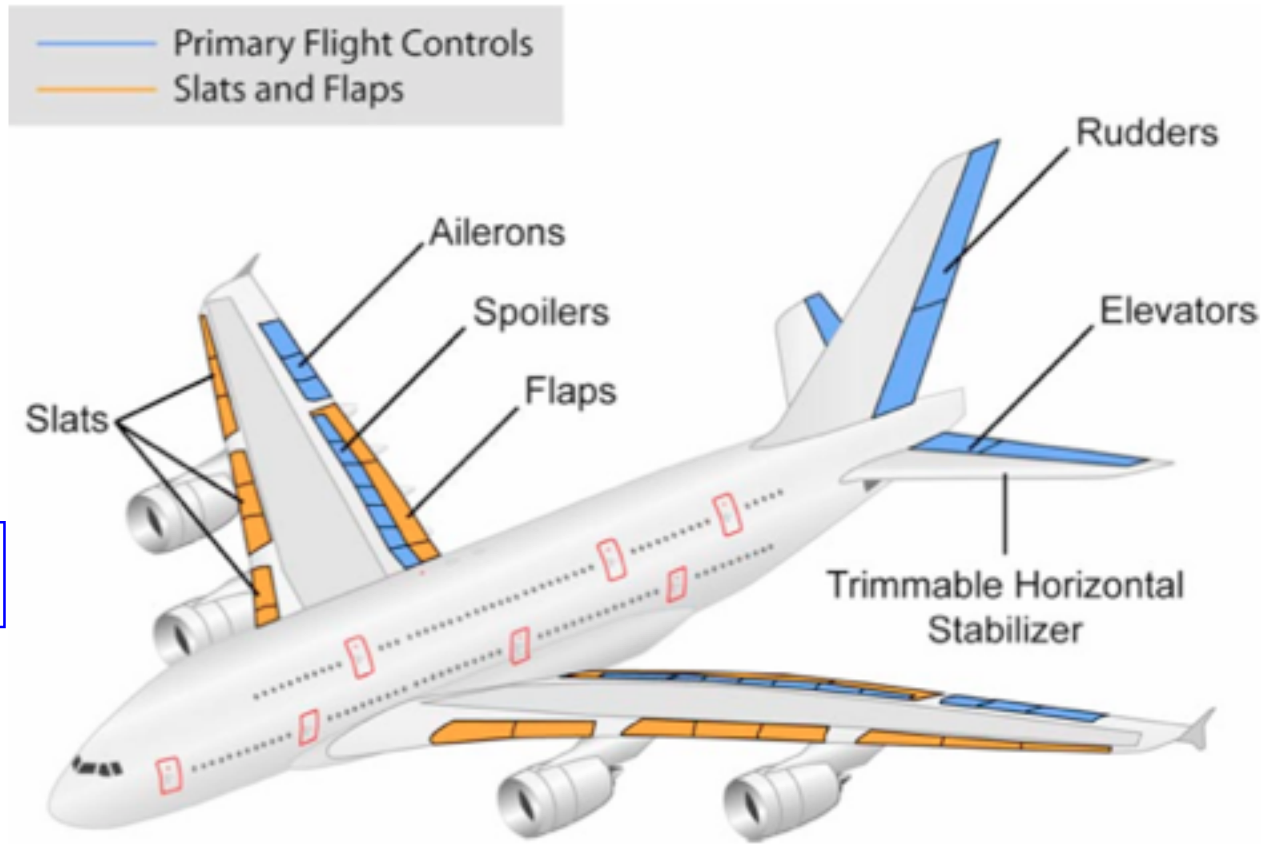
Compiling critical embedded software

Execute pilot's commands

Flight assistance: keep aircraft within safe flight envelope

Fly-by-wire software





Mostly control-command code (Scade) + a minimalistic OS (C)

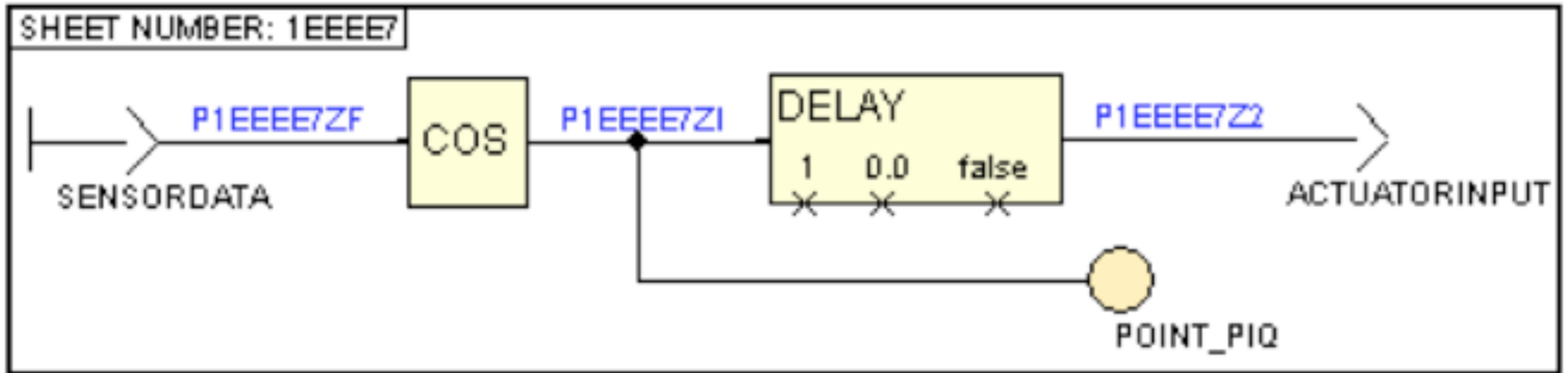
Fly-by-wire software

100k - 1M LOC code, but mostly generated from block diagrams (Simulink, Scade)

The qualification process (DO-178)

- Rigorous validation: review (qualitative), analysis (quantitative), testing (huge amounts)
- Conducted at multiple levels, from design to final product
- Meticulous development process; extensive documentation

From block diagrams to code



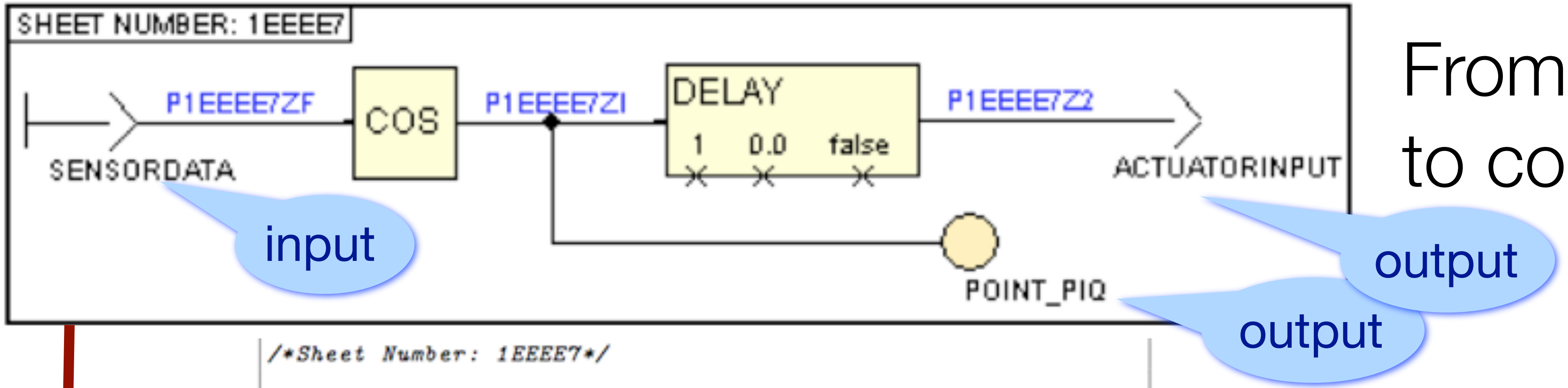
code generator

```
/*Sheet Number: 1EEEE7*/

#include "other_includes.h"
#include "delay.mac"
#include "cos.mac"
#include "piq_x.mac"

m_START_NUM_INPUT_ZONE
m_INPUT(SENSORDATA,NUM)
m_END_NUM_INPUT_ZONE

m_START_NUM_OUTPUT_ZONE
m_OUTPUT(ACTUATORINPUT,NUM)
m_END_NUM_OUTPUT_ZONE

m_START_NUM_OBS_ZONE
m_OBS_NUM(POINT_PIQ)
m_END_NUM_OBS_ZONE

m_START_CST_INPUT_ZONE
m_CST(c_DELAY_P3_2_N1EEEE7,ENT, 1)
m_END_CST_INPUT_ZONE

m_START_STATIC_ARRAY_ZONE
m_STATIC_ARRAY(t_DELAY_2_N1EEEE7, 1,NUM)
m_END_STATIC_ARRAY_ZONE

m_START_SHEET(1E,EE,E7)

m_CONNECTION(P1EEEE7Z2_N1EEEE7,NUM)
m_CONNECTION(loc_c_DELAY_P3_2_N1EEEE7,ENT)
m_CONNECTION(P1EEEE7ZF_N1EEEE7,NUM)
m_CONNECTION(P1EEEE7ZI_N1EEEE7,NUM)

m_VtS(SENSORDATA,P1EEEE7ZF_N1EEEE7)
m_CtS(c_DELAY_P3_2_N1EEEE7,loc_c_DELAY_P3_2_N1EEEE7)

COS(0_N1EEEE7,P1EEEE7ZF_N1EEEE7,P1EEEE7ZI_N1EEEE7)
PIQ_X(1_N1EEEE7,P1EEEE7ZI_N1EEEE7,POINT_PIQ)
DELAY(2_N1EEEE7,P1EEEE7ZI_N1EEEE7,\
loc_c_DELAY_P3_2_N1EEEE7,P1EEEE7Z2_N1EEEE7,t_DELAY_2_N1EEEE7)

m_StV(P1EEEE7Z2_N1EEEE7,ACTUATORINPUT)
m_END_SHEET
```

compiler

```
   Nm_1EEEE7:
   ; annotation: Symbol DELAY number 2_N1EEEE7 ,\
 inputs: f3, r31 and one static
100 addis      r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
104 lwz        r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
   ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
   ; annotation: DELAY; is entered with r4 = from 0 to
108 mr         r7, r4
10c addis      r12, 0, (t_DELAY_2_N1EEEE7)@ha
110 addi       r8, r12, (t_DELAY_2_N1EEEE7)@l
114 rlwinm     r10, r7, 3, 0, 28 ; 0xfffffff8
118 add        r10, r8, r10
11c lfd        f2, 0(r10)
   ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
   ; annotation: DELAY; is entered with r4 = from 0 to
120 mr         r8, r4
124 addis      r12, 0, (t_DELAY_2_N1EEEE7)@ha
128 addi       r6, r12, (t_DELAY_2_N1EEEE7)@l
12c rlwinm     r5, r8, 3, 0, 28 ; 0xfffffff8
130 add        r9, r6, r5
134 stfd       f3, 0(r9)
138 addi       r4, r4, 1
13c cmpw       cr0, r4, r31
140 bt         0, .L101
144 addi       r4, 0, 0
   .L101:
148 addis      r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
14c stw r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
   ; annotation: End of DELAY number 2_N1EEEE7 , output: f2
```

From block diagrams to code



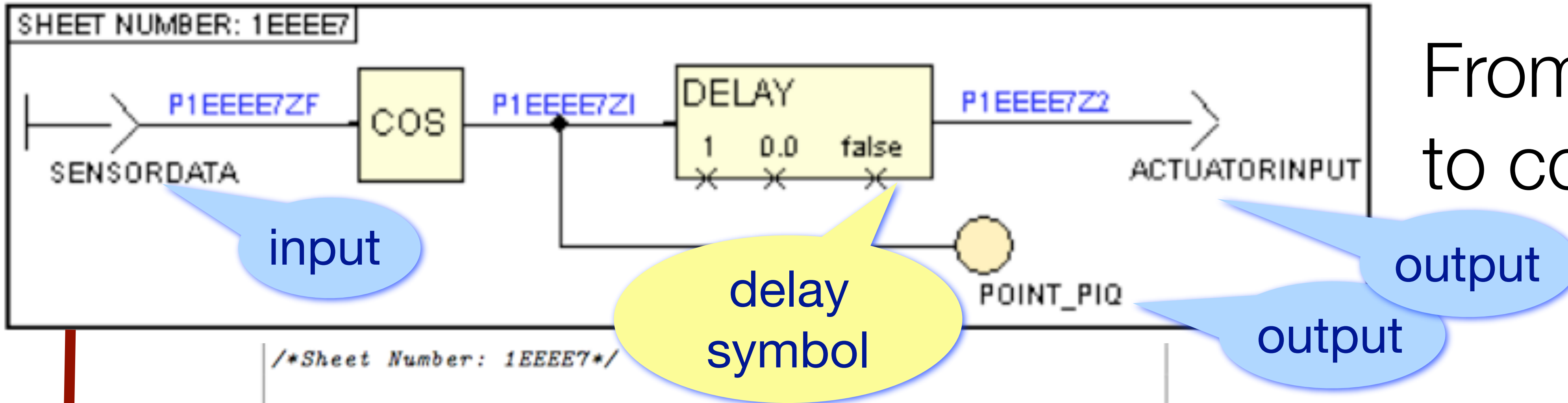
code generator

```
/*Sheet Number: 1EEEE7*/

#include "other_includes.h"
#include "delay.mac"
#include "cos.mac"
#include "piq_x.mac"

m_START_NUM_INPUT_ZONE
m_INPUT(SENSORDATA,NUM)
m_END_NUM_INPUT_ZONE

m_START_NUM_OUTPUT_ZONE
m_OUTPUT(ACTUATORINPUT,NUM)
m_END_NUM_OUTPUT_ZONE

m_START_NUM_OBS_ZONE
m_OBS_NUM(POINT_PIQ)
m_END_NUM_OBS_ZONE

m_START_CST_INPUT_ZONE
m_CST(c_DELAY_P3_2_N1EEEE7,ENT, 1)
m_END_CST_INPUT_ZONE

m_START_STATIC_ARRAY_ZONE
m_STATIC_ARRAY(t_DELAY_2_N1EEEE7, 1,NUM)
m_END_STATIC_ARRAY_ZONE

m_START_SHEET(1E,EE,E7)

m_CONNECTION(P1EEEE7Z2_N1EEEE7,NUM)
m_CONNECTION(loc_c_DELAY_P3_2_N1EEEE7,ENT)
m_CONNECTION(P1EEEE7ZF_N1EEEE7,NUM)
m_CONNECTION(P1EEEE7ZI_N1EEEE7,NUM)

m_VtS(SENSORDATA,P1EEEE7ZF_N1EEEE7)
m_CtS(c_DELAY_P3_2_N1EEEE7,loc_c_DELAY_P3_2_N1EEEE7)

COS(0_N1EEEE7,P1EEEE7ZF_N1EEEE7,P1EEEE7ZI_N1EEEE7)
PIQ_X(1_N1EEEE7,P1EEEE7ZI_N1EEEE7,POINT_PIQ)
DELAY(2_N1EEEE7,P1EEEE7ZI_N1EEEE7,\
loc_c_DELAY_P3_2_N1EEEE7,P1EEEE7Z2_N1EEEE7,t_DELAY_2_N1EEEE7)

m_StV(P1EEEE7Z2_N1EEEE7,ACTUATORINPUT)
m_END_SHEET
```

compiler

```
    Nm_1EEEE7:
    ; annotation: Symbol DELAY number 2_N1EEEE7 ,\
 inputs: f3, r31 and one static
100 addis        r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
104 lwz          r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
108 mr           r7, r4
10c addis        r12, 0, (t_DELAY_2_N1EEEE7)@ha
110 addi         r8, r12, (t_DELAY_2_N1EEEE7)@l
114 rlwinm       r10, r7, 3, 0, 28 ; 0xfffffff8
118 add          r10, r8, r10
11c lfd          f2, 0(r10)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
120 mr           r8, r4
124 addis        r12, 0, (t_DELAY_2_N1EEEE7)@ha
128 addi         r6, r12, (t_DELAY_2_N1EEEE7)@l
12c rlwinm       r5, r8, 3, 0, 28 ; 0xfffffff8
130 add          r9, r6, r5
134 stfd         f3, 0(r9)
138 addi         r4, r4, 1
13c cmpw         cr0, r4, r31
140 bt           0, .L101
144 addi         r4, 0, 0
    .L101:
148 addis        r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
14c stw r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: End of DELAY number 2_N1EEEE7 , output: f2
```

From block diagrams to code



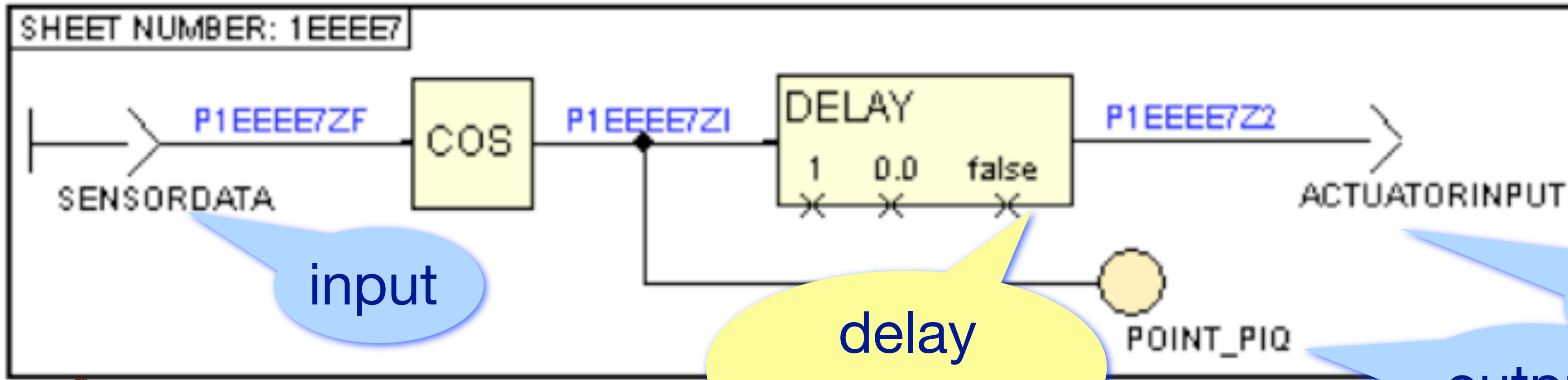
code generator

```
/*Sheet Number: 1EEEE7*/

#include "other_includes.h"
#include "delay.mac"
#include "cos.mac"
#include "piq_x.mac"

m_START_NUM_INPUT_ZONE
m_INPUT(SENSORDATA,NUM)
m_END_NUM_INPUT_ZONE

m_START_NUM_OUTPUT_ZONE
m_OUTPUT(ACTUATORINPUT,NUM)
m_END_NUM_OUTPUT_ZONE

m_START_NUM_OBS_ZONE
m_OBS_NUM(POINT_PIQ)
m_END_NUM_OBS_ZONE

m_START_CST_INPUT_ZONE
m_CST(c_DELAY_P3_2_N1EEEE7,ENT, 1)
m_END_CST_INPUT_ZONE

m_START_STATIC_ARRAY_ZONE
m_STATIC_ARRAY(t_DELAY_2_N1EEEE7, 1,NUM)
m_END_STATIC_ARRAY_ZONE

m_START_SHEET(1E,EE,E7)

m_CONNECTION(P1EEEE7Z2_N1EEEE7,NUM)
m_CONNECTION(loc_c_DELAY_P3_2_N1EEEE7,ENT)
m_CONNECTION(P1EEEE7ZF_N1EEEE7,NUM)
m_CONNECTION(P1EEEE7ZI_N1EEEE7,NUM)

m_VtS(SENSORDATA,P1EEEE7ZF_N1EEEE7)
m_CtS(c_DELAY_P3_2_N1EEEE7,loc_c_DELAY_P3_2_N1EEEE7)

COS(0_N1EEEE7,P1EEEE7ZF_N1EEEE7,P1EEEE7ZI_N1EEEE7)
PIQ_X(1_N1EEEE7,P1EEEE7ZI_N1EEEE7,POINT_PIQ)
DELAY(2_N1EEEE7,P1EEEE7ZI_N1EEEE7,\
loc_c_DELAY_P3_2_N1EEEE7,P1EEEE7Z2_N1EEEE7,t_DELAY_2_N1EEEE7)

m_StV(P1EEEE7Z2_N1EEEE7,ACTUATORINPUT)
m_END_SHEET
```

compiler

```
    Nm_1EEEE7:
    ; annotation: Symbol DELAY number 2_N1EEEE7 ,\
 inputs: f3, r31 and one static
100 addis      r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
104 lwz        r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
108 mr         r7, r4
10c addis      r12, 0, (t_DELAY_2_N1EEEE7)@ha
110 addi       r8, r12, (t_DELAY_2_N1EEEE7)@l
114 rlwinm     r10, r7, 3, 0, 28 ; 0xfffffff8
118 add        r10, r8, r10
11c lfd        f2, 0(r10)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
120 mr         r8, r4
124 addis      r12, 0, (t_DELAY_2_N1EEEE7)@ha
128 addi       r6, r12, (t_DELAY_2_N1EEEE7)@l
12c rlwinm     r5, r8, 3, 0, 28 ; 0xfffffff8
130 add        r9, r6, r5
134 stfd       f3, 0(r9)
138 addi       r4, r4, 1
13c cmpw       cr0, r4, r31
140 bt         0, .L101
144 addi       r4, 0, 0
    .L101:
148 addis      r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
14c stw r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: End of DELAY number 2_N1EEEE7 , output: f2
```

From block diagrams to code



code generator

delay macro

```
/*Sheet Number: 1EEEE7*/

#include "other_includes.h"
#include "delay.mac"
#include "cos.mac"
#include "piq_x.mac"

m_START_NUM_INPUT_ZONE
m_INPUT(SENSORDATA,NUM)
m_END_NUM_INPUT_ZONE

m_START_NUM_OUTPUT_ZONE
m_OUTPUT(ACTUATORINPUT,NUM)
m_END_NUM_OUTPUT_ZONE

m_START_NUM_OBS_ZONE
m_OBS_NUM(POINT_PIQ)
m_END_NUM_OBS_ZONE

m_START_CST_INPUT_ZONE
m_CST(c_DELAY_P3_2_N1EEEE7,ENT, 1)
m_END_CST_INPUT_ZONE

m_START_STATIC_ARRAY_ZONE
m_STATIC_ARRAY(t_DELAY_2_N1EEEE7, 1,NUM)
m_END_STATIC_ARRAY_ZONE

m_START_SHEET(1E,EE,E7)

m_CONNECTION(P1EEEE7Z2_N1EEEE7,NUM)
m_CONNECTION(loc_c_DELAY_P3_2_N1EEEE7,ENT)
m_CONNECTION(P1EEEE7ZF_N1EEEE7,NUM)
m_CONNECTION(P1EEEE7ZI_N1EEEE7,NUM)

m_VtS(SENSORDATA,P1EEEE7ZF_N1EEEE7)
m_CtS(c_DELAY_P3_2_N1EEEE7,loc_c_DELAY_P3_2_N1EEEE7)

COS(0_N1EEEE7,P1EEEE7ZF_N1EEEE7,P1EEEE7ZI_N1EEEE7)
PIQ_X(1_N1EEEE7,P1EEEE7ZI_N1EEEE7,POINT_PIQ)
DELAY(2_N1EEEE7,P1EEEE7ZI_N1EEEE7,\
loc_c_DELAY_P3_2_N1EEEE7,P1EEEE7Z2_N1EEEE7,t_DELAY_2_N1EEEE7)

m_StV(P1EEEE7Z2_N1EEEE7,ACTUATORINPUT)
m_END_SHEET
```

delay macro

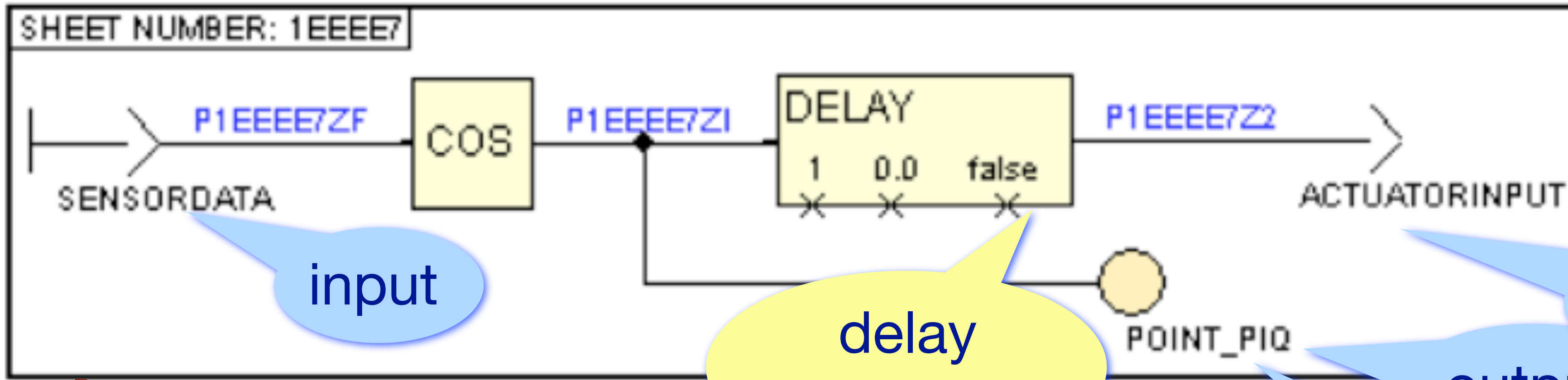
compiler

delay symbol

```
    Nm_1EEEE7:
    ; annotation: Symbol DELAY number 2_N1EEEE7 ,\
 inputs: f3, r31 and one static
100 addis      r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
104 lwz        r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
108 mr         r7, r4
10c addis      r12, 0, (t_DELAY_2_N1EEEE7)@ha
110 addi       r8, r12, (t_DELAY_2_N1EEEE7)@l
114 rlwinm     r10, r7, 3, 0, 28 ; 0xfffffff8
118 add        r10, r8, r10
11c lfd        f2, 0(r10)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
120 mr         r8, r4
124 addis      r12, 0, (t_DELAY_2_N1EEEE7)@ha
128 addi       r6, r12, (t_DELAY_2_N1EEEE7)@l
12c rlwinm     r5, r8, 3, 0, 28 ; 0xfffffff8
130 add        r9, r6, r5
134 stfd       f3, 0(r9)
138 addi       r4, r4, 1
13c cmpw       cr0, r4, r31
140 bt         0, .L101
144 addi       r4, 0, 0
    .L101:
148 addis      r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
14c stw r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: End of DELAY number 2_N1EEEE7 , output: f2
```

delay symbol

From block diagrams to code



code generator

delay macro

variable stored in RAM

compiler

```
/*Sheet Number: 1EEEE7*/

#include "other_includes.h"
#include "delay.mac"
#include "cos.mac"
#include "piq_x.mac"

m_START_NUM_INPUT_ZONE
m_INPUT(SENSORDATA,NUM)
m_END_NUM_INPUT_ZONE

m_START_NUM_OUTPUT_ZONE
m_OUTPUT(ACTUATORINPUT,NUM)
m_END_NUM_OUTPUT_ZONE

m_START_NUM_OBS_ZONE
m_OBS_NUM(POINT_PIQ)
m_END_NUM_OBS_ZONE

m_START_CST_INPUT_ZONE
m_CST(c_DELAY_P3_2_N1EEEE7,ENT, 1)
m_END_CST_INPUT_ZONE

m_START_STATIC_ARRAY_ZONE
m_STATIC_ARRAY(t_DELAY_2_N1EEEE7, 1,NUM)
m_END_STATIC_ARRAY_ZONE

m_START_SHEET(1E,EE,E7)

m_CONNECTION(P1EEEE7Z2_N1EEEE7,NUM)
m_CONNECTION(loc_c_DELAY_P3_2_N1EEEE7,ENT)
m_CONNECTION(P1EEEE7ZF_N1EEEE7,NUM)
m_CONNECTION(P1EEEE7ZI_N1EEEE7,NUM)

m_VtS(SENSORDATA,P1EEEE7ZF_N1EEEE7)
m_CtS(c_DELAY_P3_2_N1EEEE7,loc_c_DELAY_P3_2_N1EEEE7)

COS(0_N1EEEE7,P1EEEE7ZF_N1EEEE7,P1EEEE7ZI_N1EEEE7)
PIQ_X(1_N1EEEE7,P1EEEE7ZI_N1EEEE7,POINT_PIQ)
DELAY(2_N1EEEE7,P1EEEE7ZI_N1EEEE7,\
loc_c_DELAY_P3_2_N1EEEE7,P1EEEE7Z2_N1EEEE7,t_DELAY_2_N1EEEE7)

m_StV(P1EEEE7Z2_N1EEEE7,ACTUATORINPUT)
m_END_SHEET
```

delay macro

delay symbol

```
    Nm_1EEEE7:
    ; annotation: Symbol DELAY number 2_N1EEEE7 ,\
 inputs: f3, r31 and one static
100 addis       r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
104 lwz         r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
108 mr          r7, r4
10c addis       r12, 0, (t_DELAY_2_N1EEEE7)@ha
110 addi        r8, r12, (t_DELAY_2_N1EEEE7)@l
114 rlwinm      r10, r7, 3, 0, 28 ; 0xfffffff8
118 add         r10, r8, r10
11c lfd         f2, 0(r10)
    ; annotation: Variable to search: loc_c_DELAY_P3_2_N1EEEE7
    ; annotation: DELAY; is entered with r4 = from 0 to
120 mr          r8, r4
124 addis       r12, 0, (t_DELAY_2_N1EEEE7)@ha
128 addi        r6, r12, (t_DELAY_2_N1EEEE7)@l
12c rlwinm      r5, r8, 3, 0, 28 ; 0xfffffff8
130 add         r9, r6, r5
134 stfd        f3, 0(r9)
138 addi        r4, r4, 1
13c cmpw        cr0, r4, r31
140 bt          0, .L101
144 addi        r4, 0, 0
    .L101:
148 addis       r12, 0, (_DELAY_2_N1EEEE7_R2)@ha
14c stw r4, (_DELAY_2_N1EEEE7_R2)@l(r12)
    ; annotation: End of DELAY number 2_N1EEEE7 , output: f2
```

delay symbol

Program annotations

A mechanism to attach annotations to program points

- Mark specific program points
- Provide information about the location of C variables.
- Ensure that some variables are preserved (e.g. x must be kept in a register).

Annotations are preserved during compilation.

- Each annotation generates an observable event.

```
_annot("Begin loop");
...
x = 1;
_annot("Here x is in %1",x);
...
_annot("End loop");
```

compiler

```
; annotation: Begin loop
...
addi r3, 0, 1
; annotation: Here x is in r3
...
; annotation: End loop
```

Conformance to the qualification process

A formally verified compiler gives traceability guarantees.

Simplified example

- The semantics preservation theorem ensures preservation of:
  - the sequencing of symbols,
  - the sequencing of accesses to hardware devices (volatile variables).

Remember the main theorem: If the source program can not go wrong, then the behavior of the generated assembly code is exactly one of the behaviors of the source program.

```
Theorem transf_c_program_is_refinement:
forall p tp, transf_c_program p = OK tp →
(forall behv, exec_C_program p behv → not_wrong behv) →
(forall behv,  exec_Asm_program tp behv → exec_C_program p behv).
```

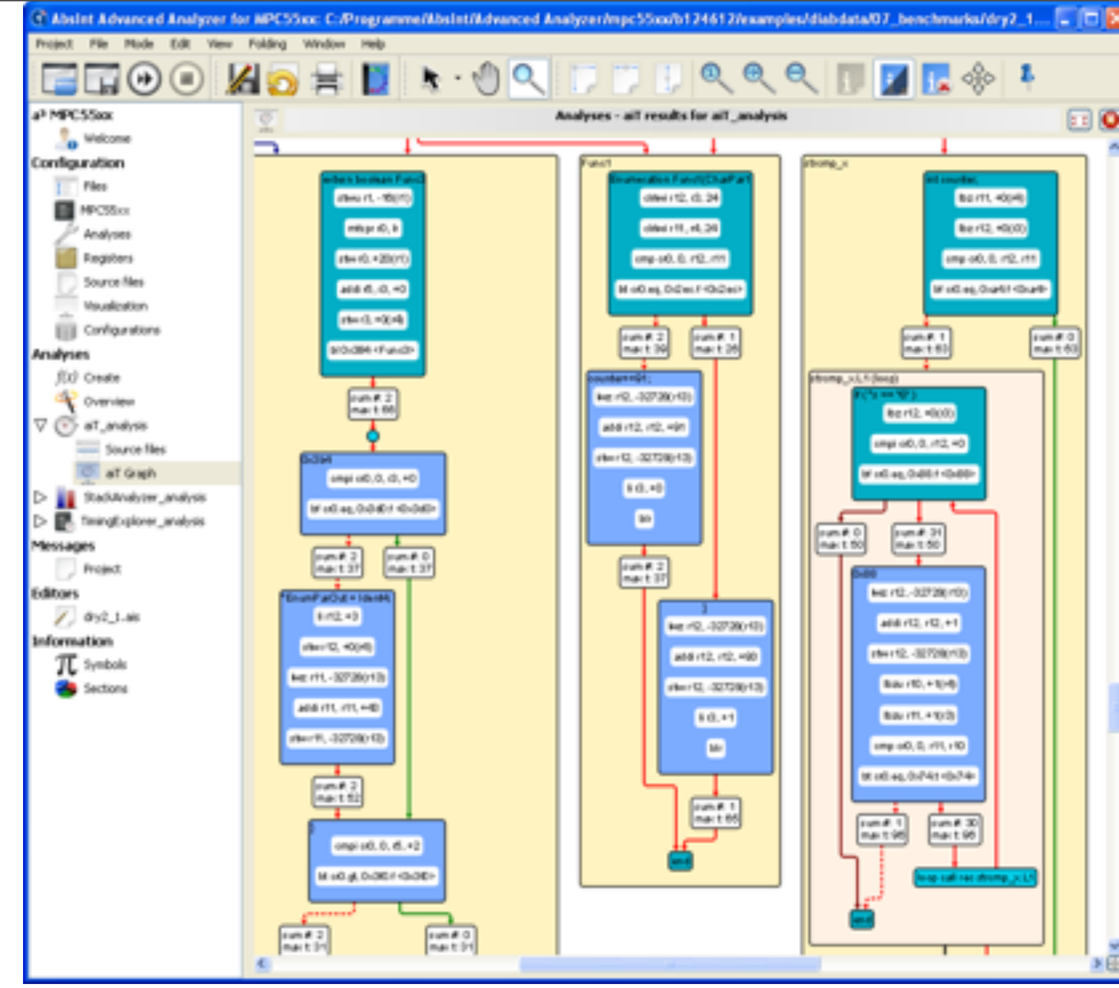
How good is the compiled code ?

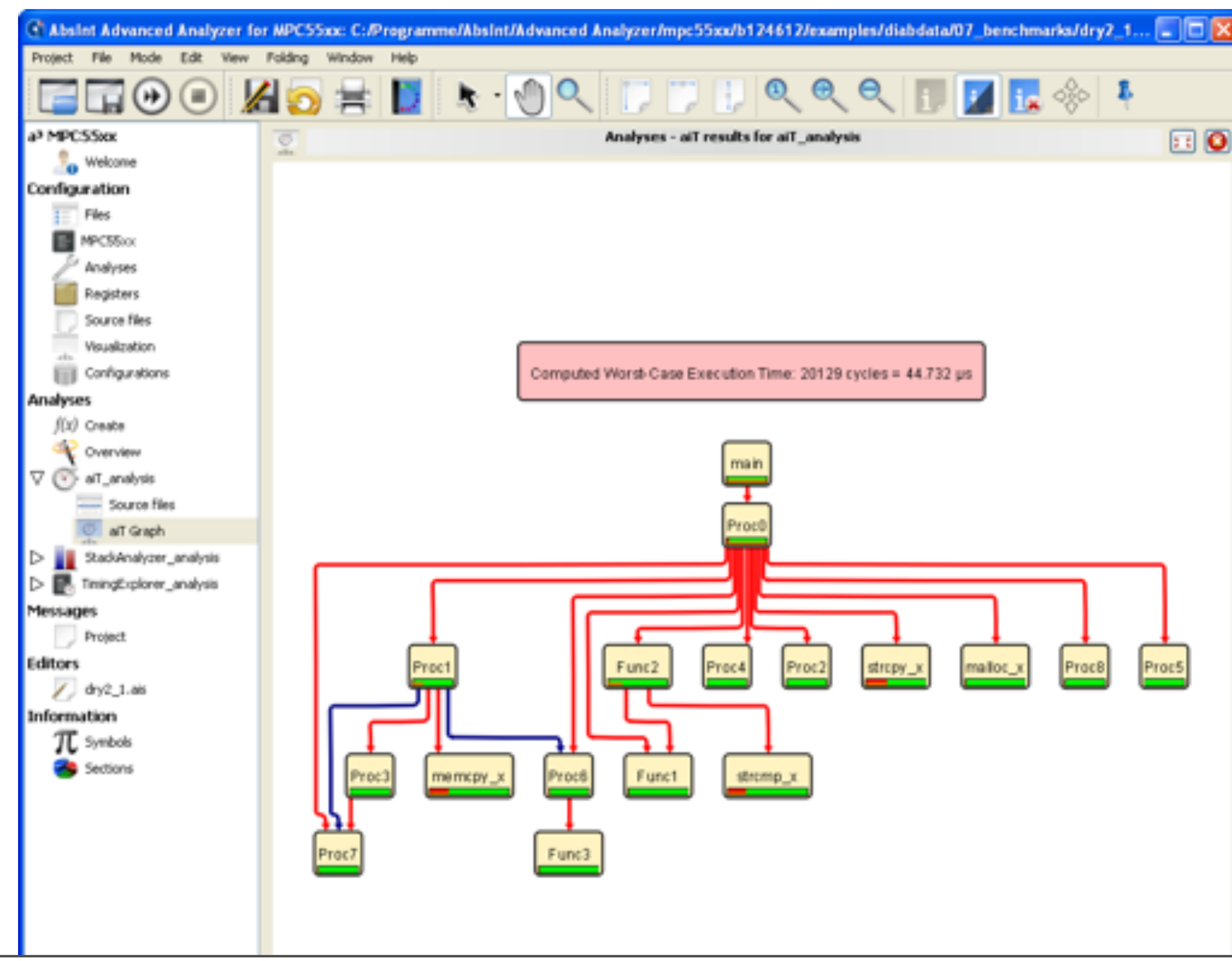
Trade-off between

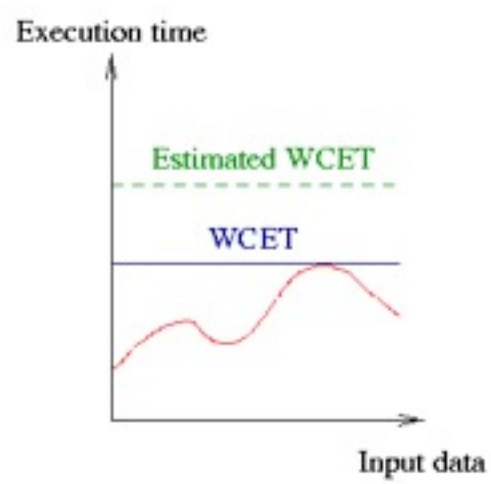
- traceability guarantees
- and efficiency of the generated code

Low-level verifications

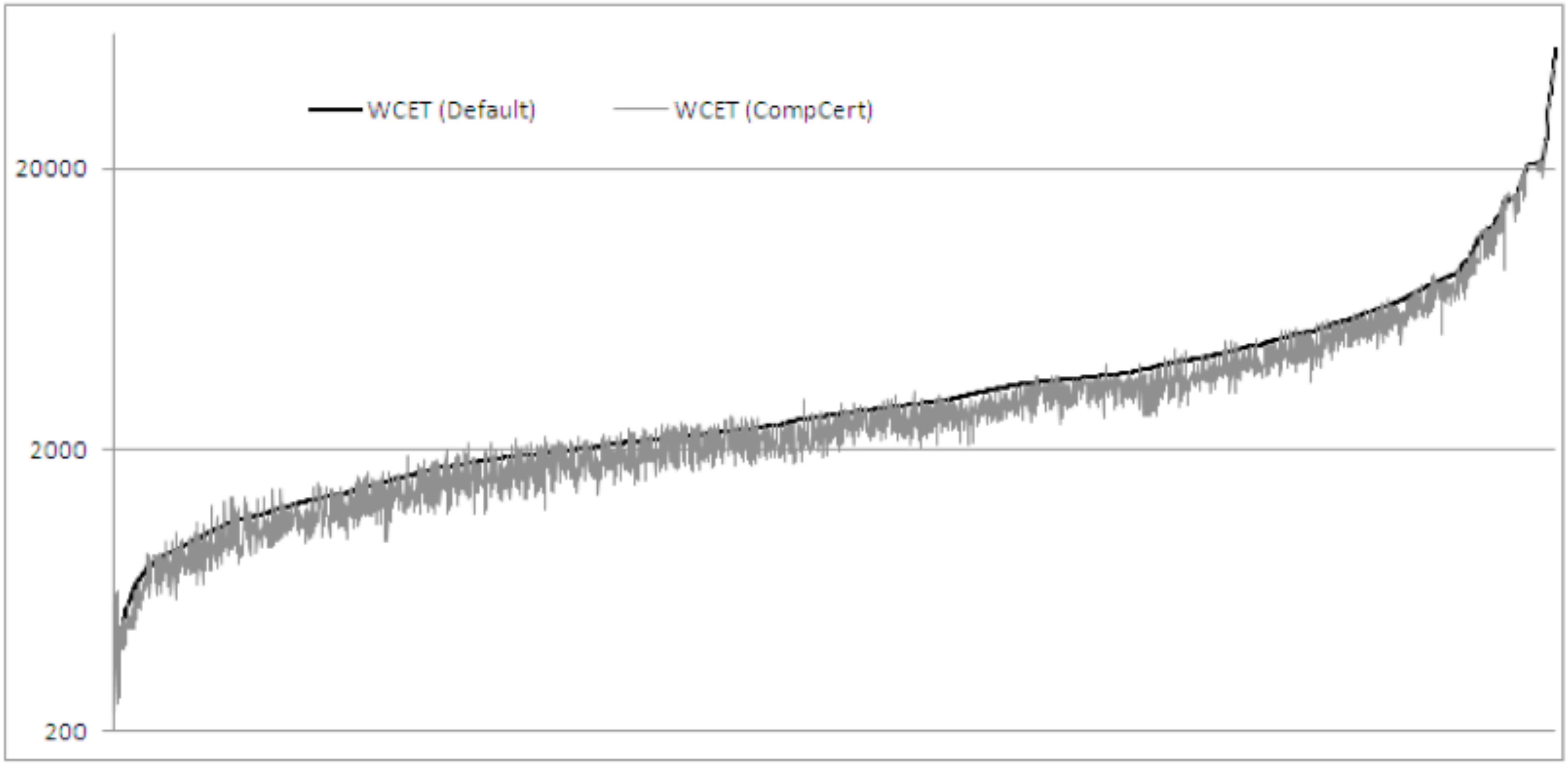
- reviews of the assembly
- computation of a WCET estimation







WCET improvement



FCGU A380: 3600 files, 3.96 MB of assembly code

- Estimated WCET for each file
- Average improvement per file: 13,5%
- Compiled with CompCert 1.10, March 2012

Overall assessment

The improvement mainly comes from the register allocation pass.

- From: no register allocation
- To: sharing of local variables among available registers

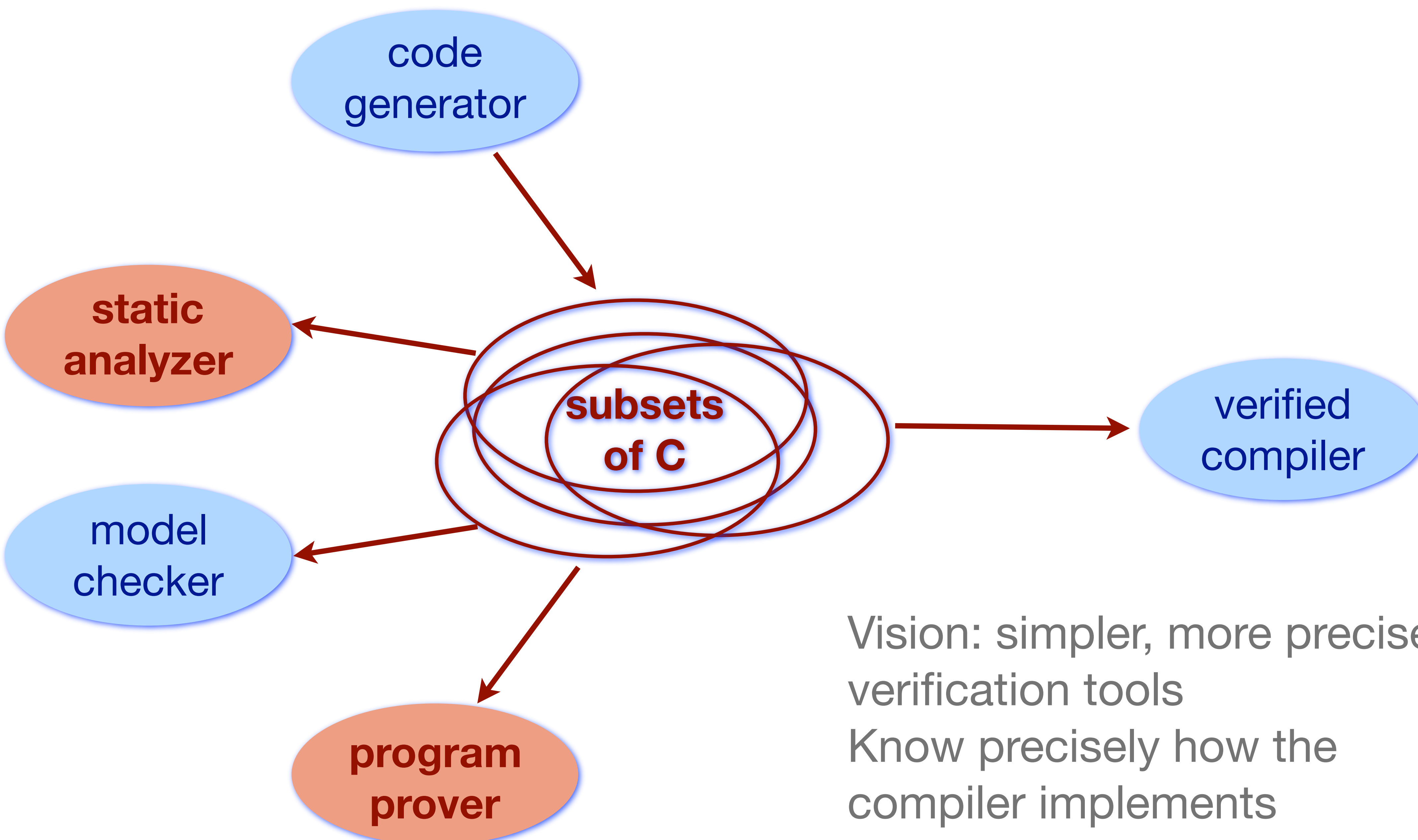
Traceability guarantees

- From: tracking of all program variables
- To: tracking of meaningful variables (existing in block diagrams)

Concluding remarks
Reusable libraries

Concluding remarks
Connections with verification tools

Are these verification tools semantically sound ?



Vision: simpler, more precise verification tools
Know precisely how the compiler implements unspecified behaviors of C

Concluding remarks
Higher assurance

Towards a qualification strategy for CompCert (DO-178C)

- How to qualify a C compiler ? a Coq formal development ?
- CompCert is specified by the semantics of its source and target languages (incl. supporting theories: machine integers, floats, I/O model and memory model), and by the semantics preservation theorem
  - Use of interpreters to test these semantics
  - Thanks to the proof, no need to talk about intermediate languages, compilation algorithms, optimizations and their supporting static analyses.
- Trust in Coq's extraction ? Trust in Caml ?