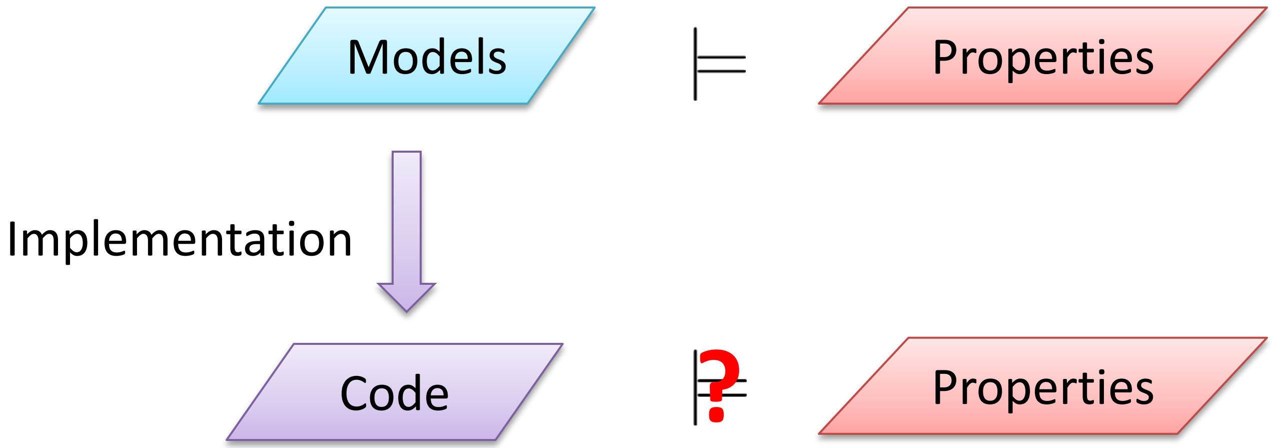
Automatic Generation of Provably Correct Embedded Systems

Shang-Wei LIN, Yang LIU, Pao-Ann HSIUNG, Jun SUN, and Jin Song DONG

Temasek Laboratories
National University of Singapore

ICFEM 2012, Kyoto Japan

Motivation

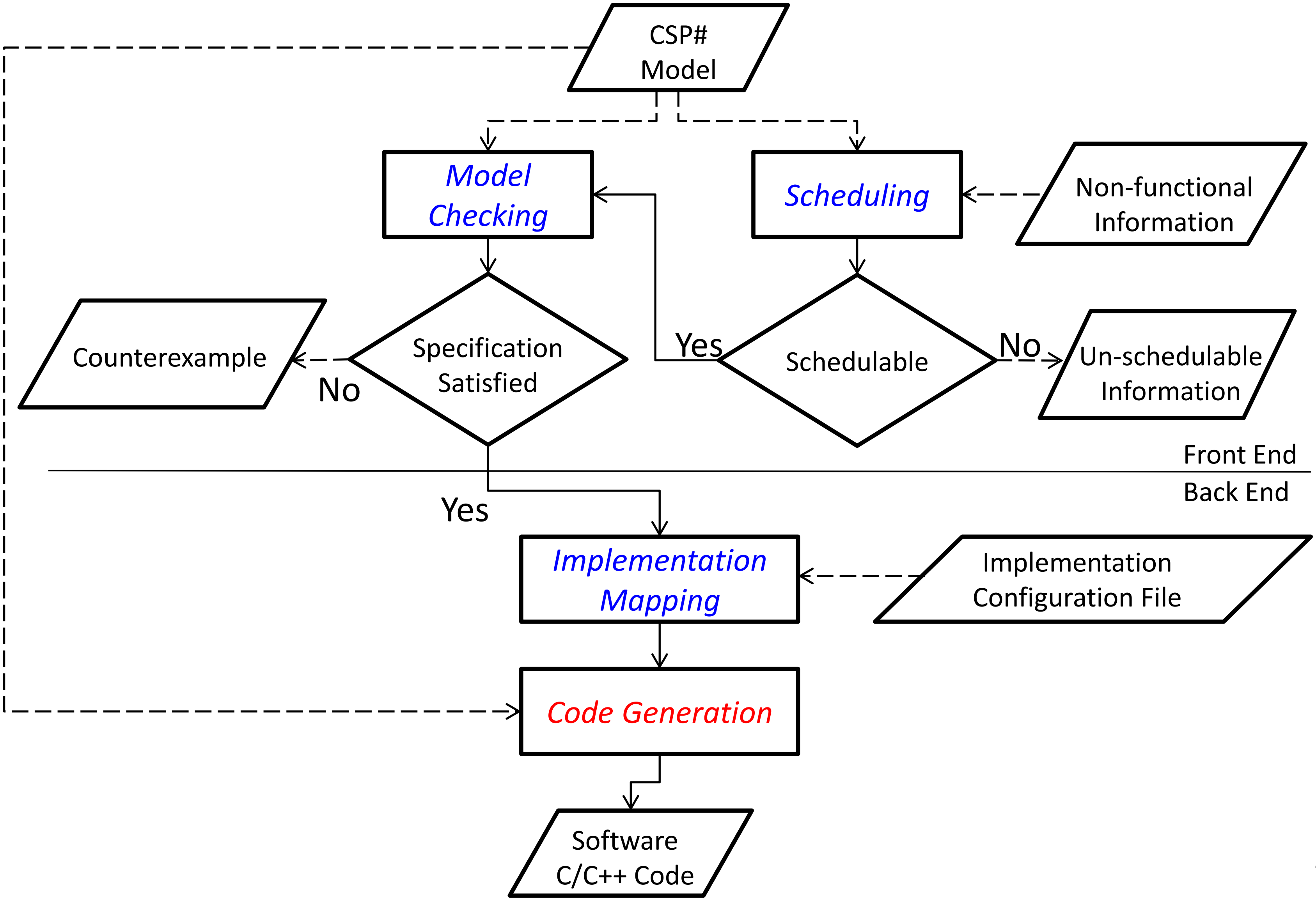


- Can we generate code *automatically*?
  - Verification results are still valid

Outline

- Overall Flow
- Code Generation Approach
- Example
- Case Studies
- Future Work

Overall Flow



Overall Flow (cont.)

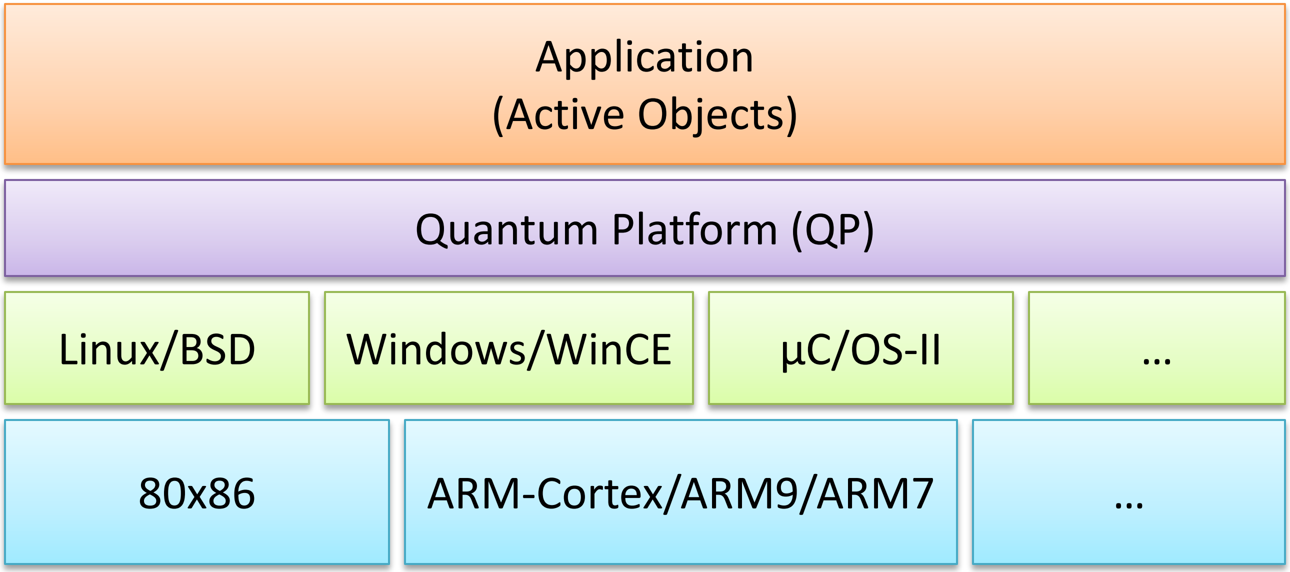
- Input
  - *Communicating Sequential Program* (CSP#)
    - Extension of *Communicating Sequential Process* (CSP)
    - Message passing and shared memory
    - Declaration of variables
    - Data operations on declared variables
- Output
  - C/C++ software code

Outline

- Overall Flow
- Code Generation Approach
- Example
- Case Studies
- Future Work

Code Generation Approach

- CSP# → state machines → QP active objects
  - Generated code is **OS-independent** and **hardware-independent**



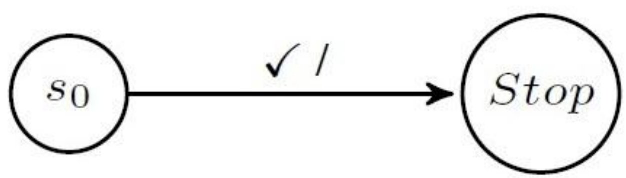
CSP# to State Machines

- Translation rule for each primitive CSP# process
  - *Stop*
  - Skip: √ → *Stop*
  - prefix: *e* { *prog* } → *P*
  - invariant: [*b*] *P*
  - conditional choice: if *b* { *P* } else { *Q* }
  - channel input: *ch*?x → *P*
  - channel output: *ch*!x → *P*
  - interleaving: *P* ||| *Q*
  - sequential: *P* ; *Q*
  - general choice: *P* [] *Q*

Translation Rules

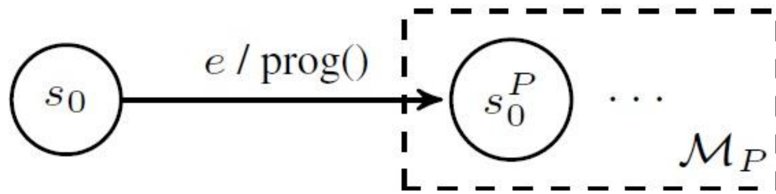
- *Stop*

- *Skip*
  - √ → *Stop*

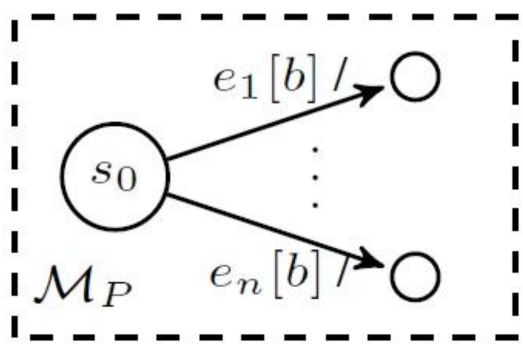


Translation Rules (cont.)

- *e* { *prog* } → *P*

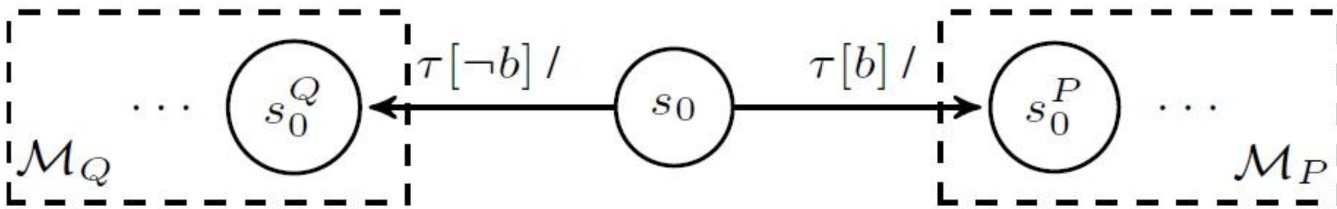


- [*b*] *P*



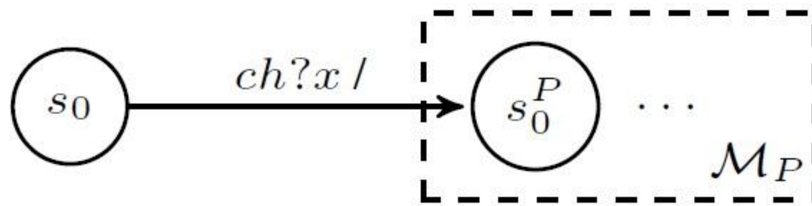
Translation Rules (cont.)

- if b { P } else { Q }

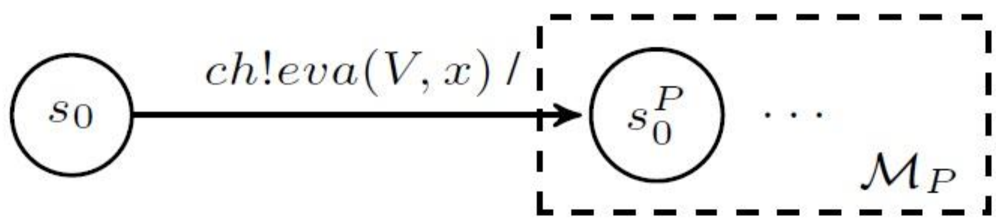


Translation Rules (cont.)

- *ch*?*x* → *P*

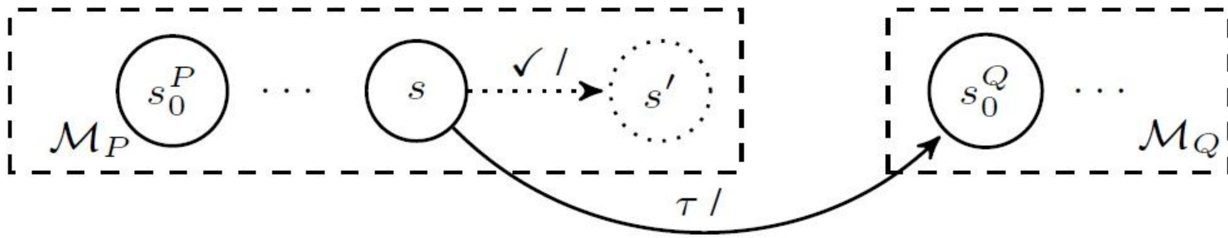


- *ch*!*x* → *P*

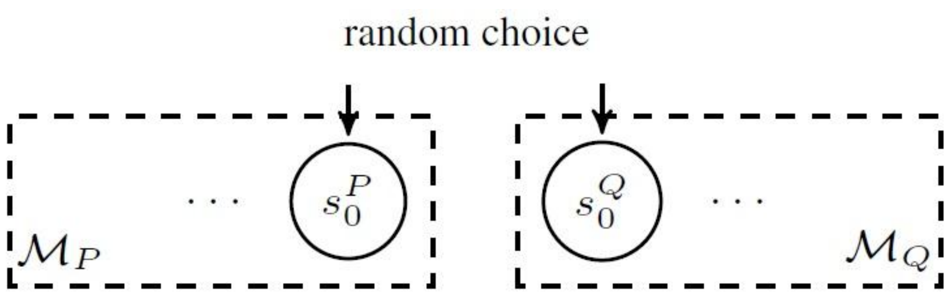


Translation Rules (cont.)

- *P*; *Q*



- *P* [] *Q*



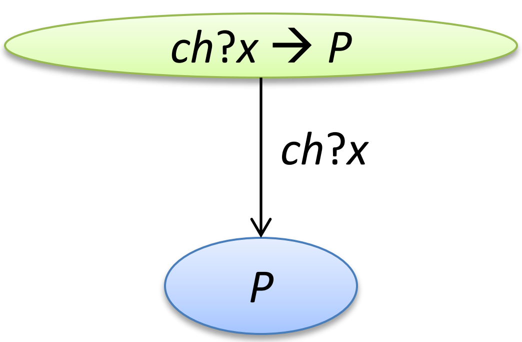
Correctness

Theorem 1.

The translated state machine is a bisimulation of the original CSP# model

[proof idea]

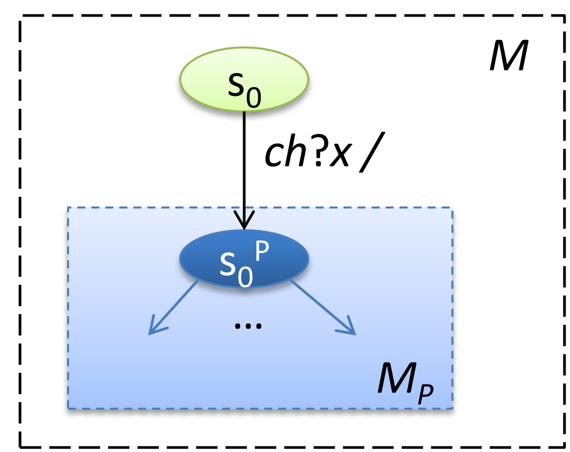
- The translated state machine for each primitive process is a bisimulation.
- A CSP# process is composed with several primitive processes inductively.
- For example, $ch?x \rightarrow P$



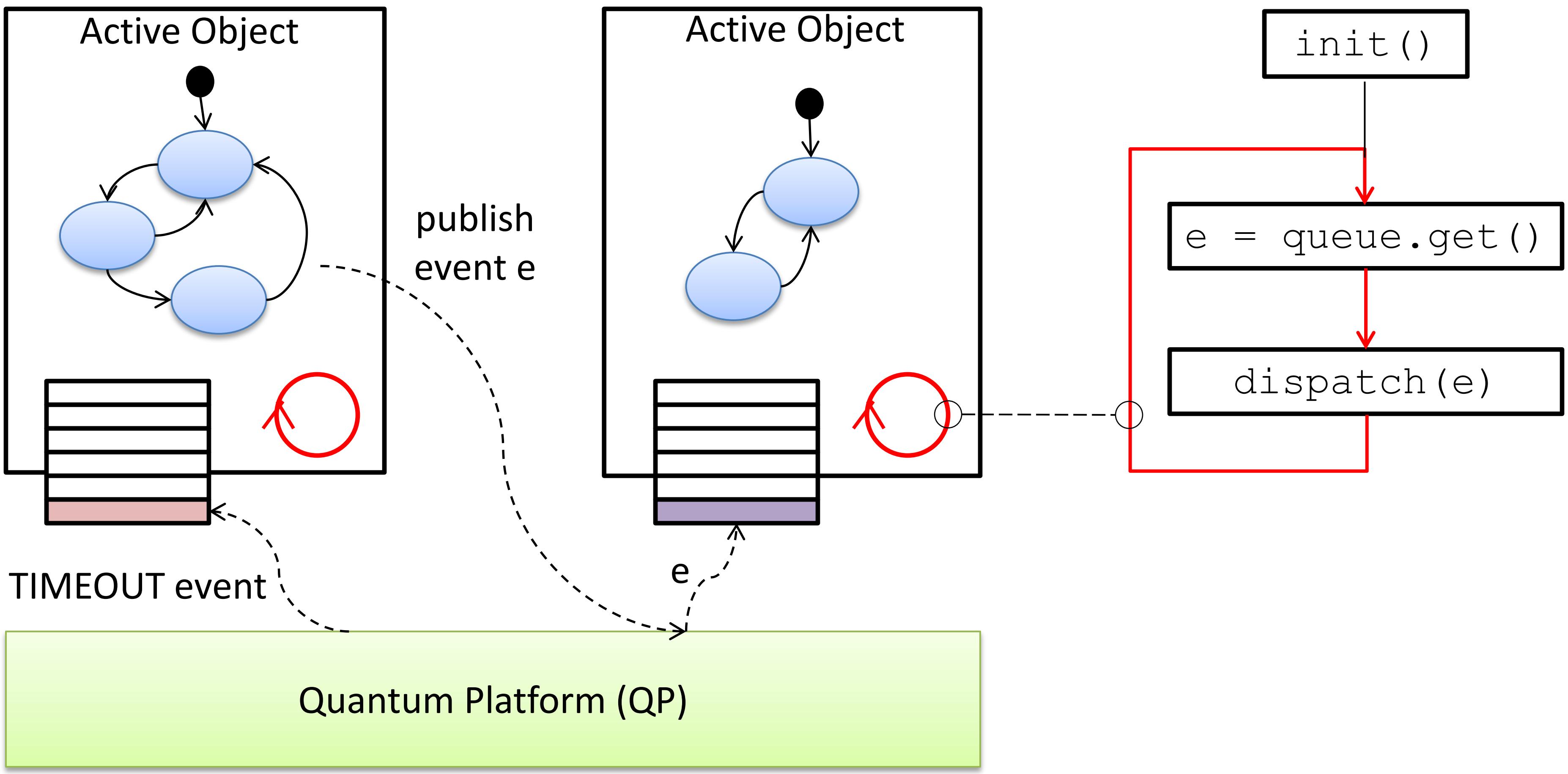
$P \approx M_P$

$\downarrow$

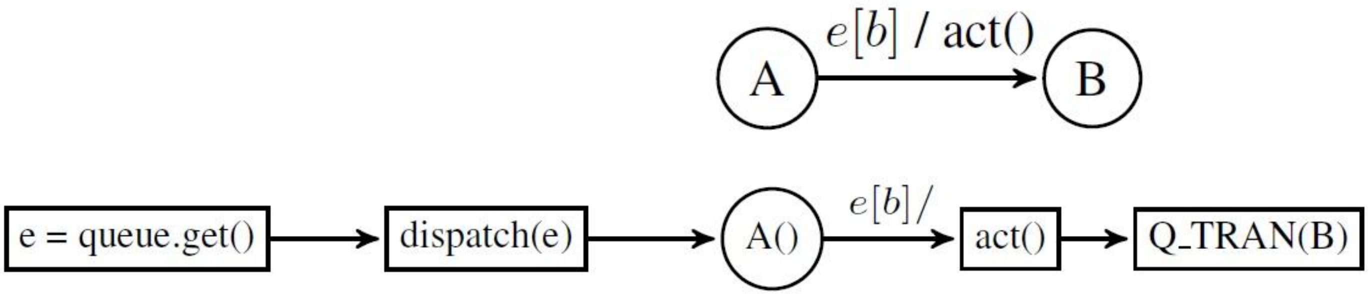
$ch?x \rightarrow P \approx M$



State Machines to C/C++ Code



State Machines to C/C++ Code (cont.)



Theorem 2.

The behavior of the implementation in active objects conforms to the original state machines.

[proof idea]

Each transition in an active object conforms to the operational semantics of its corresponding state machine

Overall Correctness

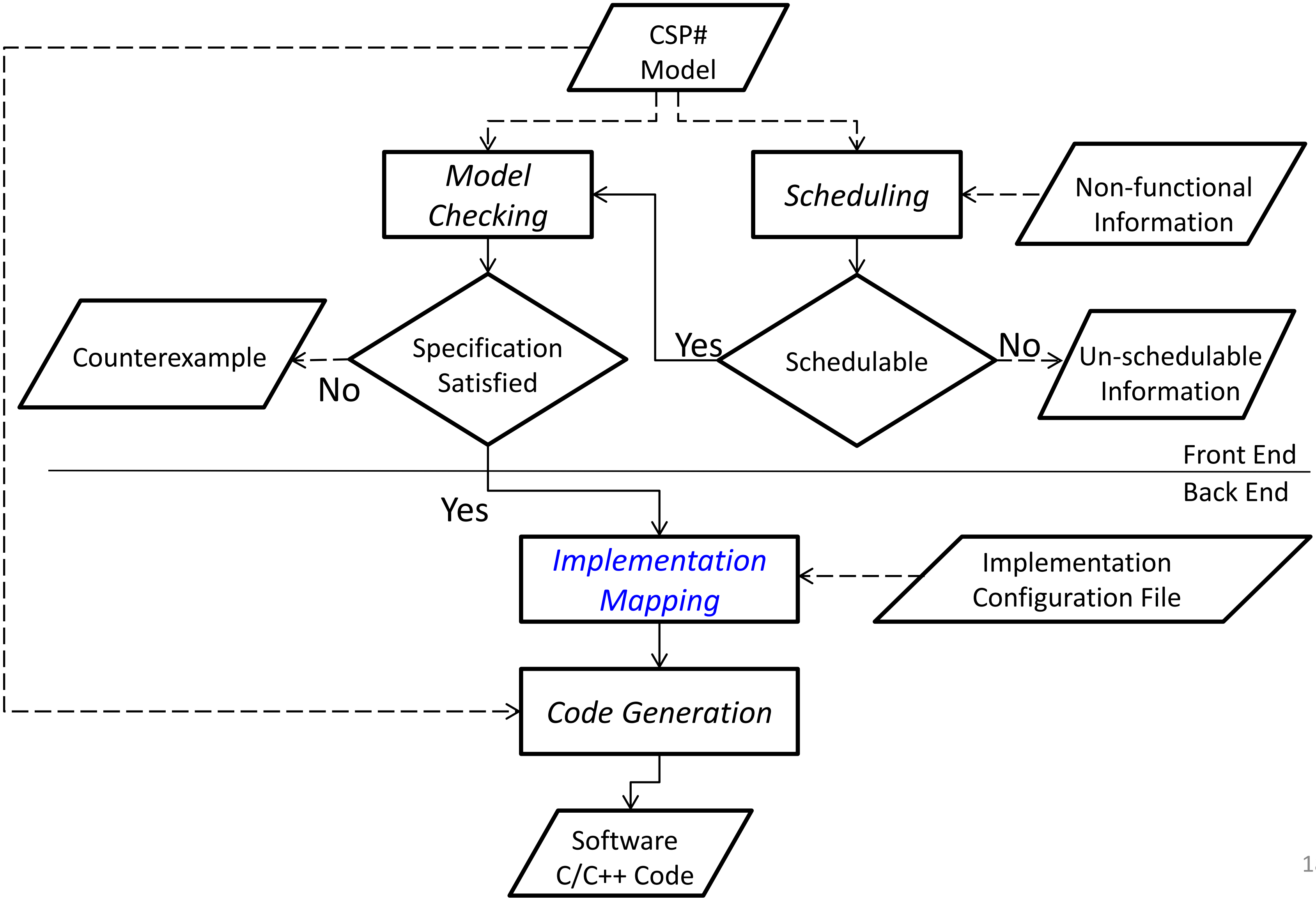
CSP# Processes ↔ state machines ↔ active objects

Theorem 1 Theorem 2

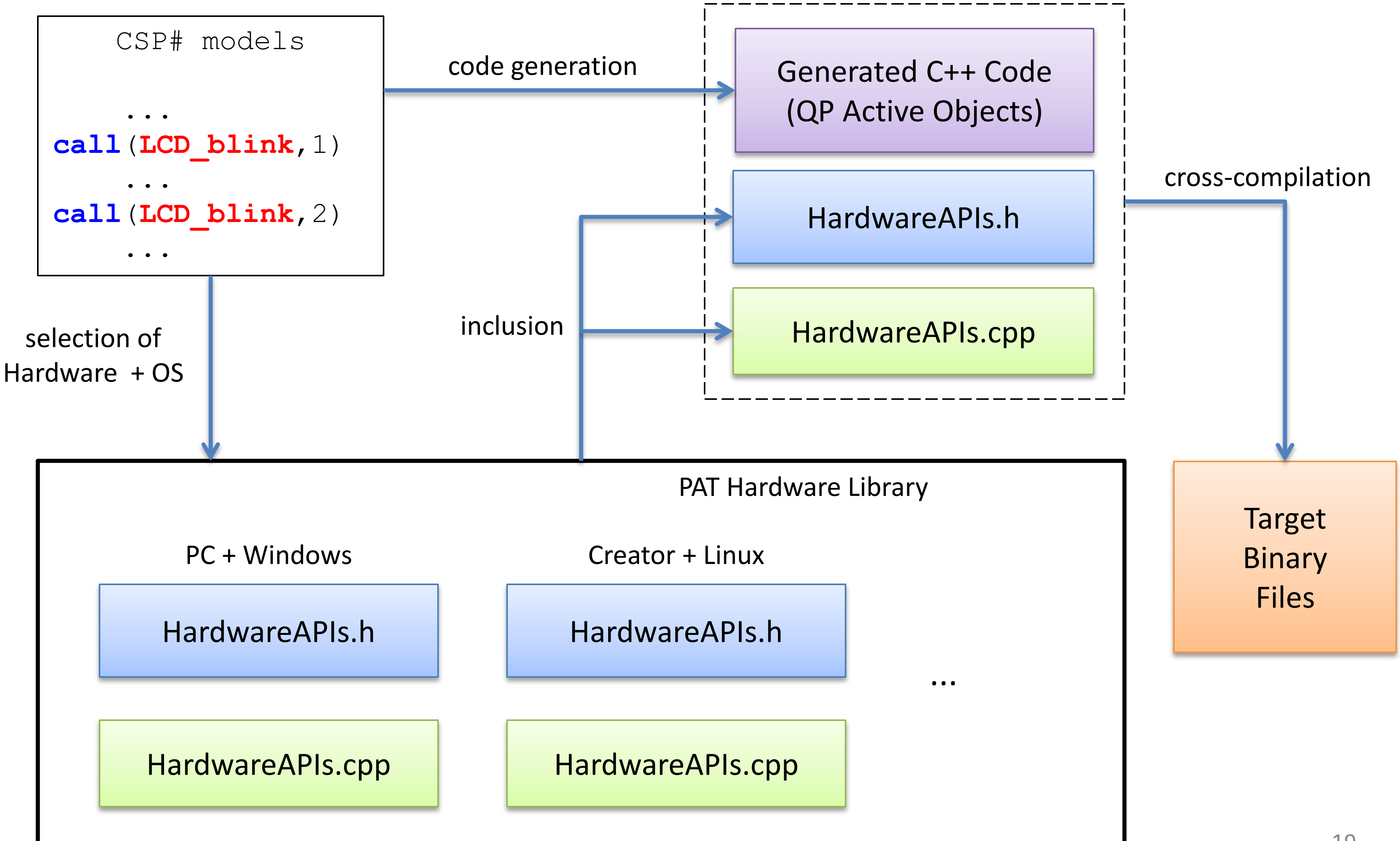
[Assumption]
The execution of the action on a transition of an active object is **atomic**

[Discharge]
synchronization using mutexes on the execution of the action

Generic Hardware APIs



Generic Hardware APIs



Outline

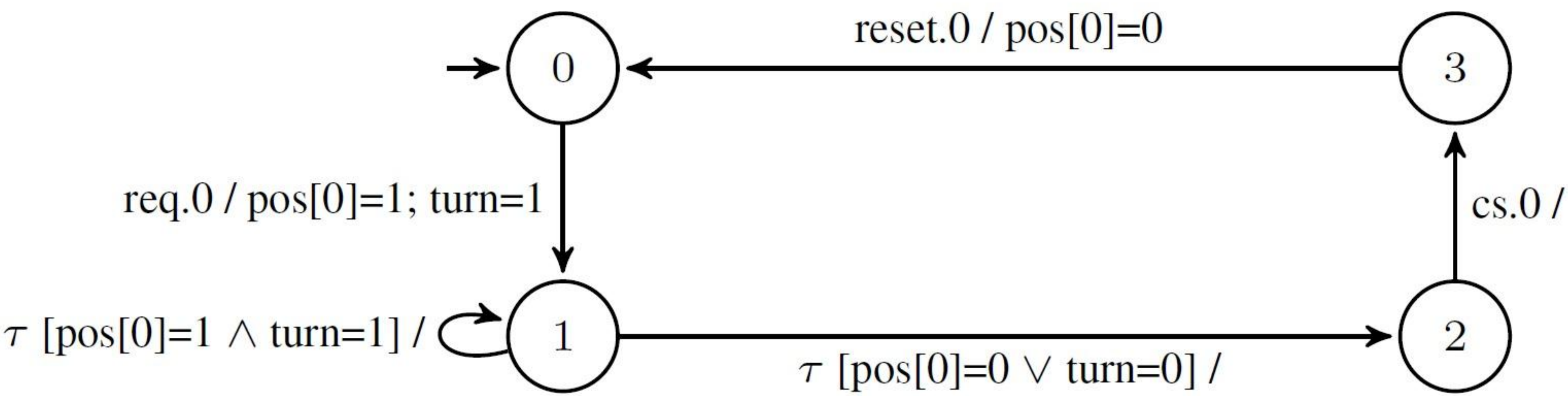
- Overall Flow
- Code Generation Approach
- Example
- Case Studies
- Future Work

Peterson's Protocol

```
#define N 2;
var turn;
var pos[N];

P0() = req.0{pos[0] = 1; turn=1} ->
       Wait0(); cs.0 -> reset.0{pos[0] = 0} -> P0();
Wait0() = if (pos[1]==1 && turn == 1) { Wait0() };


P1() = req.1{pos[1] = 1; turn=0} ->
       Wait1(); cs.1 -> reset.1{pos[1] = 0} -> P1();
Wait1() = if (pos[0]==1 && turn == 0) { Wait1() };


Peterson() = P0() ||| P1();
```

Peterson's Protocol (cont.)

Translated state machine for process P_0



Peterson's Protocol (cont.)

```
class P0 : public QActive {                                /** P0.h  **/
   public:
      P0();
      ~P0();

   private:
      static QState initial(P0 *me, QEvent const *e);
      static QState P0_0(P0 *me, QEvent const *e);
      static QState P0_1(P0 *me, QEvent const *e);
      static QState P0_2(P0 *me, QEvent const *e);
      static QState P0_3(P0 *me, QEvent const *e);

   public:
      void req_0();
      void cs_0();
      void reset_0();

   private:
      QTimeEvt m_timeEvt;
};
```

Peterson’s Protocol (cont.)

```
P0::P0() :                                    /** P0.cpp  **/
QActive((QStateHandler)&P0::initial),m_timeEvt(TIMEOUT_SIG){}

P0::~P0(){}

QState P0::initial(P0 *me, QEvent const *){
   return Q_TRAN(&P0::P0_0);
}

void P0::req_0(){
   std::cout<<"req_0"<<std::endl;
}

void P0::cs_0(){
   std::cout<<"cs_0"<<std::endl;
}

void P0::reset_0(){
   std::cout<<"reset_0"<<std::endl;
}
```

Peterson's Protocol (cont.)

```
QState P0::                                   /** P0.cpp (cont.)**/
P0_0(P0 *me, QEvent const *e){
   QEvent *pe;

   switch(e->sig){
      case Q_ENTRY_SIG:
         me->m_timeEvt.postIn(me, WAIT_TIME);
         return Q_HANDLED();

      case Q_EXIT_SIG:
         return Q_HANDLED();

      case TIMEOUT_SIG:
         me->req_0();
         pos[0]=1;turn=1;
         return Q_TRAN(&P0::P0_1);
   }

   return Q_SUPER(&QHsm::top);
}
```

Peterson's Protocol (cont.)

```
QState P0::                                   /** P0.cpp (cont.)**/
P0_1(P0 *me, QEvent const *e){
   QEvent *pe;

   switch(e->sig){
      case Q_ENTRY_SIG:
         me->m_timeEvt.postIn(me, WAIT_TIME);
         return Q_HANDLED();

      case Q_EXIT_SIG:
         return Q_HANDLED();

      case TIMEOUT_SIG:
         if(((pos[1] == 1) && (turn == 1))){
            return Q_TRAN(&P0::P0_1);
         }
         if(!(((pos[1] == 1) && (turn == 1)))){
            return Q_TRAN(&P0::P0_2);
         }
   }
   return Q_SUPER(&QHsm::top);
}
```

Peterson's Protocol (cont.)

```
QState P0::
P0_2(P0 *me, QEvent const *e){
   QEvent *pe;

   switch(e->sig){
      case Q_ENTRY_SIG:
         me->m_timeEvt.postIn(me, WAIT_TIME);
         return Q_HANDLED();

      case Q_EXIT_SIG:
         return Q_HANDLED();

      case TIMEOUT_SIG:
         me->cs_0();
         return Q_TRAN(&P0::P0_3);
   }
   return Q_SUPER(&QHsm::top);
}
```

Peterson's Protocol (cont.)

```
QState P0::
P0_3(P0 *me, QEvent const *e){
   QEvent *pe;

   switch(e->sig){
      case Q_ENTRY_SIG:
         me->m_timeEvt.postIn(me, WAIT_TIME);
         return Q_HANDLED();

      case Q_EXIT_SIG:
         return Q_HANDLED();

      case TIMEOUT_SIG:
         me->reset_0();
         pos[0]=0;
         return Q_TRAN(&P0::P0_0);
   }
   return Q_SUPER(&QHsm::top);
}
```

Peterson's Protocol (cont.)

```
        // Main.cpp

static P0 __P0;
static P1 __P1;

static QEvent const *l_P0_QueueSto[10];
static QEvent const *l_P1_QueueSto[10];

static QSubscrList   l_subscrSto[MAX_PUB_SIG];

static union SmallEvents {
   void *e0;
   uint8_t e1[sizeof(DataEvent)];
       // other event types to go into this pool
} l_smlPoolSto[2 * (1 + 0) * 2];
      // storage for the small event pool
              …
```

Peterson's Protocol (cont.)

```
int main(int argc, char *argv[]) {

   QF::init();
   QF::psInit(l_subscrSto, Q_DIM(l_subscrSto));
   QF::poolInit(l_smlPoolSto, sizeof(l_smlPoolSto),
                sizeof(l_smlPoolSto[0]));

   __P0.start((uint8_t)(1), l_tableQueueSto,
              Q_DIM(l_tableQueueSto),
              (void *)0, 0, (QEvent *)0);

   __P1.start((uint8_t)(2), l_tableQueueSto,
              Q_DIM(l_tableQueueSto),
              (void *)0, 0, (QEvent *)0);

   QF::run();
   return 0;
}
```

Outline

- Overall Flow
- Code Generation Approach
- Example
- Case Studies
- Future Work

Case Studies

- Entrance Guard System (EGS)
  - controls the entrance of a building
  - Input, Display, Controller, DBMS, Actuator, Alarm

- Secure Communication Box (SCB)
  - provides a secure communication between two clients

Example - Secure Communication Box

```
#define USER_CONNECT 77;     #define SERVER_READY 78;
#define DATA 79;             #define POWER_ON 80;

channel chAdmin 5;      channel chA 5;         channel chB 5;

Admin() = chAdmin!POWER_ON -> Skip;

User1() = chA!USER_CONNECT -> chA?SERVER_READY -> SendData1();
SendData1() = chA!DATA -> send_data_1 -> ReceiveData1();
ReceiveData1() = chA?x -> receive_data_1 -> SendData1();

User2() = chB!USER_CONNECT -> chB?SERVER_READY -> ReceiveData2();
ReceiveData2() = chB?x -> receive_data_2 -> SendData2();
SendData2() = chB!DATA -> send_data_2 -> ReceiveData2();


Server() = chAdmin?POWER_ON -> power_up {call(LCD_blink,1)}
          -> initializatioin -> User1Connected();
User1Connected() = chA?USER_CONNECT {call(LCD_blink,2)}
                  -> chA!SERVER_READY -> User2Connected();
User2Connected() = chB?USER_CONNECT {call(LCD_blink,3)}
                  -> chB!SERVER_READY -> transimitData();
transimitData() = chA?x {call(LCD_blink,4)} -> chB!x {call(LCD_blink,5)} -> transimitData();

System = User1() ||| Server() ||| User2() ||| Admin();
```

Outline

- Overall Flow
- Code Generation Approach
- Example
- Case Studies
- Future Work

Future Work

- Automatic Code Generation for
  - Real-time Systems
  - Multi-core Platforms