A hardware/software co-design flow for the SynDEVS MoC was presented in this thesis. The SynDEVS MoC was introduced which is a revised version of the time-discrete DEVS formalism from Zeigler et al. It was reviewed in detail in terms of synthesis aspects in contrast to the non-synthesizable DEVS MoC. Understanding the behaviour of SynDEVS models was boosted by declaring a visual representation in favour of the more formal pure mathematical notation. This visual programming paradigm was further extended by developing a GUI which allows for the creation of models. By doing so, all intermediate models of the co-design flow may be reviewed from within the GUI.

The advocated abstract system level design flow, which allows for transformation of a such a timed MoC into into synthesizable VHDL source code and embeddable C++ source code, is the main contribution of this thesis. Both transformation paths from the SynDEVS MoC to hardware and software instances were discussed in detail. A state optimization algorithm for the DEVS MoC’s zero-timeout states, which is an inherent property of the DEVS MoC to model conditional execution within a single step in time, was introduced. A dedicated application example, the DVI controller, enabled the demonstration of the important steps within the design flow in order to relax the timing requirements of the synthesised model. Besides that, this illustrative example shows the effectiveness of the hardware transformation methodology regarding synthesis of a model with strict timing-requirements.

To allow an efficient use of the hardware/software co-design, the communication in between the hardware and software parts were automatically generated in terms of a proof-of-concept FSL interface link. It was emphasized that a successful introduction of MoCs into the development of embedded systems requires the possibility to include legacy software
libraries. A dedicated SynDEVS component was introduced to allow the reuse of such legacy software libraries written in C / C++ source code. Hardware/software co-design including automated communication interface generation and legacy software inclusion was demonstrated by the means of a network-based Pong game.

However, as the development of embedded system is in general an evolutionary process, it was demonstrated that the introduction of SynDEVS MoC into an existing design is feasible: An existing hardware/software co-design of a cryptographic accelerator was enriched with a SynDEVS model for evaluation of the cryptographic main operations. Results exemplify that such a late introduction of SynDEVS MoC is highly beneficial for the figures of merit in terms of throughput.

To enable the application of the advocated design flow in further areas, it was detailed how other MoCs may be transformed into the SynDEVS MoC. Then, models from these MoCs may be exploited within the advocated system level design flow, too. Furthermore, model validation in terms of simulation with SystemC was described in detail. A non-intrusive SystemC kernel extension was developed to allow a fast simulation of the SynDEVS MoC. Nevertheless, the described transformation methodology for the hardware/software co-design of the SynDEVS MoC as well as the SystemC implementation methodology may be applicable to other time-discrete MoCs, too.

Abstract system level design with MoCs in terms of SynDEVS MoC is still in its infancy. There are many open research challenges and interesting opportunities to broaden the scope of the advocated hardware/software co-design flow. So far, only models from time-discrete domains are covered but cyber physical systems demand continuously timed models, too. Thus, it is of utmost interest to expand the described methodology to cover these MoCs.

Another extension would be to expand the existing hardware/software co-design flow to feature more sophisticated design space exploration methods. Even more, an integration of reconfiguration (i.e. RecDEVS) into the lower abstraction levels of the design flow would be interesting and will rise important questions. For instance, how can the time flow within a model preserved if reconfiguration will occur during the execution of the model, which itself needs time.

An introduction of a special purpose processing unit for SynDEVS MoC would be another interesting aspect. Then, software execution with respect to timing could be achieved more efficiently in contrast to the exploitation
of an untimed software language like C++ which is a cumbersome task. Having that said, it is still useful because of the legacy software library support. Hence, exploitation of multi-core architectures on the software part of the advocated hardware/software co-design flow could be enhanced by using a RTOS\textsuperscript{1}. Such an approach would allow the reuse of more complex legacy software libraries including those which require operation system support (e.g. task management). By doing so, the application field of abstract system level design with MoCs would be further broadened.

\textsuperscript{1} Real-time Operating System
APPENDIX A

Source Codes

A.1 DEVS Component

Listing A.1  DEVS_Component<

```cpp
1 template <class State> class DEVS_Component_Advance_Callback;
2 template <class State> void *run_devs_internal_transition_thread(
    void *param);
3 template <class State> void *run_devs_external_transition_thread(
    void *param);
4 template <class State> void *run_devs_confluence_transition_thread(
    void *param);

5 // define the debug output granularity
6 #define DBG_NONE 0x0000
7 #define DBG_ADVANCE 0x0001
8 #define DBG_DELTA 0x0002
9 #define DBG_INT_TRANSITION 0x0004
10 #define DBG_EXT_TRANSITION 0x0008
11 #define DBG_CONFLUENCE 0x0010
12 #define DBG_TIMEOUT 0x0020

15 // some combinations of debug flags from above
16 #define DBG_ALL 0x003F
17 #define DBG_TRANSITION 0x000C

19 // optimization flags
20 #define OPT_NONE 0x0000
21 #define OPT_ALL_PARALLEL 0x0001
22 #define OPT_ALL_SERIAL 0x0002
23 #define OPT_NEEDED_SERIAL 0x0004

25 // parallel execution flags
26 #define NONE_EXECUTED 0x0000
27 #define INT_EXECUTED 0x0001
```

H. G. Molter, SynDEVS Co-Design Flow,
#define EXT_EXECUTED 0x0002
#define CON_EXECUTED 0x0004
#define ALL_EXECUTED (INT_EXECUTED|EXT_EXECUTED|CON_EXECUTED)

// pthread mutex and condition helpers
#define PRE_WAIT_FOR_COND(mutex, cond) pthread_mutex_lock(&mutex);
#define POST_WAIT_FOR_COND(mutex, cond) pthread_cond_wait(&cond, &mutex);
#define POST_WAIT_FOR_COND_EXECUTED(mutex, cond, signal) while (parallelExecutionStatus != signal) { pthread_cond_wait(&cond, &mutex); } pthread_mutex_unlock(&mutex);
#define SIGNAL_COND_EXECUTED(mutex, cond, signal) pthread_mutex_lock(&mutex); parallelExecutionStatus |= signal; pthread_cond_broadcast(&cond); pthread_mutex_unlock(&mutex);
#define SIGNAL_COND(mutex, cond) pthread_mutex_lock(&mutex); pthread_cond_broadcast(&cond); pthread_mutex_unlock(&mutex);

std::string timestamp(bool bWithDelta=false, const std::string post="");

template<class State> class DEVS_Component : public sc_module {
public:
friend class DEVS_Component_Advance_Callback<State>;
friend void *run_devs_internal_transition_thread<State>(void *param);
friend void *run_devs_external_transition_thread<State>(void *param);
friend void *run_devs_confluence_transition_thread<State>(void *param);

SC_CTOR(DEVS_Component) : debugLevel(DBG_NONE) {
    lastActive = DEVS_TIME_ZERO;
    optimizeLevel = OPT_ALL_PARALLEL;
    parallelExecutionStatus = NONE_EXECUTED;
    pthread_mutex_init(&advanceConditionMutex, NULL);
    pthread_cond_init(&advanceCondition, NULL);
    pthread_mutex_init(&returnToAdvanceConditionMutex, NULL);
    pthread_cond_init(&returnToAdvanceCondition, NULL);
}

virtual const char *devsname() const {
    return name();
}

State DBGgetCurState() {
    return curState;
}

void verbose(unsigned int verbosity=DBG_ADVANCE) {
A.1 DEVS Component

```cpp
67    debugLevel = verbosity;
68 }
69
70 void optimize(unsigned int opt=OPT_NEEDED_SERIAL) {
71    optimizeLevel = opt;
72 }
73
74 const void print(std::ostream& os) {
75    print(os, curState);
76 }
77
78 /*
79    * Startup the devs component.
80    */
81 void devs_startup() {
82    if (debugLevel != DBG_NONE)
83        cerr << devsname() << "_devs_startup()" << endl;
84    // make devs_advance sensitive to timeoutEvent and register the
85    devs_advance_options.set_sensitivity(&timeoutEvent);
86    register_input_ports();
87    // spawn the process (started immediately)
88    sc_spawn(DEVS_Component_Advance_Callback<State>(this),
89              sc_gen_unique_name(std::string(devsname()).append("_devs_advance").c_str()), &devs_advance_options);
90    if (optimizeLevel&OPT_ALL_PARALLEL) {
91        pthread_create(&internalThread, NULL,
92                        run_devs_internal_transition_thread<State>, this);
93        pthread_create(&externalThread, NULL,
94                        run_devs_external_transition_thread<State>, this);
95        pthread_create(&confluenceThread, NULL,
96                        run_devs_confluence_transition_thread<State>, this);
97    }
98 }
99
100 // Model-generic functions
101 private:
102
103    /*
104    * Do the final initialization of the devs component.
105    * It belongs directly to the devs_startup() hence it is not
called there because the DEVS_Component_Advance_Callback may
not yet be started.
106    */
107 void devs_initialize() {
108    curState = initialize();
109    devs_timeout();
110 }
111```
 void devs_confluence_transition_thread ( void ) {
 p r e _ w a i t _ f o r _ c o n d ( a d v a n c e _ c o n d i t i o n _ m u t e x , a d v a n c e _ c o n d i t i o n ) ;
 s i g n a l _ c o n d _ e x e c u t e d ( r e t u r n t o _ a d v a n c e _ c o n d i t i o n _ m u t e x ,
 r e t u r n t o _ a d v a n c e _ c o n d i t i o n , C O N _ E X E C U T E D ) ;
 f o r ( ; ; ) {
 a t t a c h e d c o m m e n t 112
 // wait for the signal to start the execution
 p o s t _ w a i t _ f o r _ c o n d ( a d v a n c e _ c o n d i t i o n _ m u t e x , a d v a n c e _ c o n d i t i o n ) ;
 a t t a c h e d c o m m e n t 114
 // execute the confluence transition function
 i f ( d e b u g _ l e v e l & & DBG_CONFLUENCE) {
 p r i n t ( c o u t ) ;
}
 c o n s t a t e = c o n f l u e n c e ( c u r s t a t e , d e v s _ e l a p s e d _ t i m e ) ;
 i f ( d e b u g _ l e v e l & & DBG_CONFLUENCE) {
 c o u t << ";->";  
 p r i n t ( c o u t , c o n s t a t e ) ;
 c o u t << N O R M A L () << e n d l ;
}
 a t t a c h e d c o m m e n t 125
 // signal the end of execution
 p r e _ w a i t _ f o r _ c o n d ( a d v a n c e _ c o n d i t i o n _ m u t e x , a d v a n c e _ c o n d i t i o n ) ;
 s i g n a l _ c o n d _ e x e c u t e d ( r e t u r n t o _ a d v a n c e _ c o n d i t i o n _ m u t e x ,
 r e t u r n t o _ a d v a n c e _ c o n d i t i o n , C O N _ E X E C U T E D ) ;
}
}
 void devs_external_transition_thread ( void ) {
 p r e _ w a i t _ f o r _ c o n d ( a d v a n c e _ c o n d i t i o n _ m u t e x , a d v a n c e _ c o n d i t i o n ) ;
 s i g n a l _ c o n d _ e x e c u t e d ( r e t u r n t o _ a d v a n c e _ c o n d i t i o n _ m u t e x ,
 r e t u r n t o _ a d v a n c e _ c o n d i t i o n , E X T _ E X E C U T E D ) ;
 f o r ( ; ; ) {
 a t t a c h e d c o m m e n t 135
 // wait for the signal to start the execution
 p o s t _ w a i t _ f o r _ c o n d ( a d v a n c e _ c o n d i t i o n _ m u t e x , a d v a n c e _ c o n d i t i o n ) ;
 a t t a c h e d c o m m e n t 137
 // execute the external transition function
 i f ( d e b u g _ l e v e l & & DBG_EXT_TRANSITION) {
 p r i n t ( c o u t ) ;
}
 e x t s t a t e = e x t e r n a l _ t r a n s i t i o n ( c u r s t a t e , d e v s _ e l a p s e d _ t i m e ) ;
 i f ( d e b u g _ l e v e l & & DBG_EXT_TRANSITION) {
 c o u t << ";->";  
 p r i n t ( c o u t , e x t s t a t e ) ;
 c o u t << N O R M A L () << e n d l ;
}
 a t t a c h e d c o m m e n t 148
 // signal the end of execution
 p r e _ w a i t _ f o r _ c o n d ( a d v a n c e _ c o n d i t i o n _ m u t e x , a d v a n c e _ c o n d i t i o n ) ;
 s i g n a l _ c o n d _ e x e c u t e d ( r e t u r n t o _ a d v a n c e _ c o n d i t i o n _ m u t e x ,
 r e t u r n t o _ a d v a n c e _ c o n d i t i o n , E X T _ E X E C U T E D ) ;
void devs_internal_transition_thread(void) {
    PRE_WAIT_FOR_COND(advanceConditionMutex, advanceCondition);
    SIGNAL_COND_EXECUTED(returnToAdvanceConditionMutex, returnToAdvanceCondition, INT_EXECUTED);
    for (;;) {
        // wait for the signal to start the execution
        POST_WAIT_FOR_COND(advanceConditionMutex, advanceCondition);
        // execute the internal transition function
        if (debugLevel&DBG_INT_TRANSITION) {
            cout << TRANSITION() << std::setw(12) << timestamp(":") << "[
            " << devsnanme() << "]_devs_internal_transition_";
            print(cout);
        }
        intState = internal_transition(curState);
        if (debugLevel&DBG_INT_TRANSITION) {
            cout << ".->.";
            print(cout, intState);
            cout << NORMAL() << endl;
        }
        // signal the end of execution
        PRE_WAIT_FOR_COND(advanceConditionMutex, advanceCondition);
        SIGNAL_COND_EXECUTED(returnToAdvanceConditionMutex, returnToAdvanceCondition, INT_EXECUTED);
    }
}

void devs_confluence_transition(void) {
    if (debugLevel&DBG_CONFLUENCE) {
        cout << TRANSITION() << std::setw(12) << timestamp(":") << "[
        " << devsnanme() << "]_devs_confluence_transition_";
        print(cout);
    }
    conState = confluence(curState, devs_elapsed_time());
    if (debugLevel&DBG_CONFLUENCE) {
        cout << ".->.";
        print(cout, conState);
        cout << NORMAL() << endl;
    }
}

void devs_external_transition(void) {
    if (debugLevel&DBG_EXT_TRANSITION) {
        cout << TRANSITION() << std::setw(12) << timestamp(":") << "[
        " << devsnanme() << "]_devs_external_transition_";
        print(cout);
    }
extState = external_transition(curState, devs_elapsed_time());
if (debugLevel&DBG_EXT_TRANSITION) {
    cout << "_;->_;" << std::endl;
    print(cout, extState);
    cout << NORMAL() << endl;
}

void devs_internal_transition(void) {
    if (debugLevel&DBG_INT_TRANSITION) {
        cout << TRANSITION() << std::setw(12) << timestamp(":_") << "[
" << devname() << "]_devs_internal_transition_;";
        print(cout);
    }
    intState = internal_transition(curState);
    if (debugLevel&DBG_INT_TRANSITION) {
        cout << "_;->_;" << std::endl;
        print(cout, intState);
        cout << NORMAL() << endl;
    }
}

void devs_advance(void) {
    State nextState;
    bool extTransitionOccured = false; // will be true if an external transition occurred
    bool intTransitionOccured = false; // will be true if an internal transition occurred
    std::list<devs_in_if*>::iterator it = listInputs.begin(); (it != listInputs.end()) && !extTransitionOccured;
    intTransitionOccured = (time_advance(curState) == devs_elapsed_time());
    // scan over all input ports and look if one got an event.
    for (it = listInputs.begin(); (it != listInputs.end()) && !extTransitionOccured; it++) {
        if (((*it)->event()))
            extTransitionOccured = true;
    }
    if (optimizeLevel&OPT_NEEDED_SERIAL) {
        // serial calls to the next state functions
        if (extTransitionOccured && intTransitionOccured) {
            devs_confluence_transition();
        } else if (extTransitionOccured && !intTransitionOccured) {
            devs_external_transition();
        } else if (intTransitionOccured && !extTransitionOccured) {
            devs_internal_transition();
        }
    } else if (optimizeLevel&OPT_ALL_PARALLEL) {
        // clear the status variable (describes which threads finished their execution)
    } else if (optimizeLevel&OPT_NEEDED_SERIAL) {
        // serial calls to the next state functions
        if (extTransitionOccured && intTransitionOccured) {
            devs_confluence_transition();
        } else if (extTransitionOccured && !intTransitionOccured) {
            devs_external_transition();
        } else if (intTransitionOccured && !extTransitionOccured) {
            devs_internal_transition();
        }
    } else if (optimizeLevel&OPT_ALL_PARALLEL) {
        // clear the status variable (describes which threads finished their execution)
    }
PRE_WAIT_FOR_COND(returnToAdvanceConditionMutex, returnToAdvanceCondition);

while (parallelExecutionStatus != ALL_EXECUTED) {
    pthread_cond_wait(&returnToAdvanceCondition, &returnToAdvanceConditionMutex);
}

parallelExecutionStatus = NONE_EXECUTED;

SIGNAL_COND(advanceConditionMutex, advanceCondition);

POST_WAIT_FOR_COND_EXECUTED(returnToAdvanceConditionMutex, returnToAdvanceCondition, ALL_EXECUTED);

else if (optimizeLevel&OPT_ALL_SERIAL) {
    unsigned int call = 0x7, bit;

    while (call != 0) {
        do {
            bit = 1 << (rand() %3);
        } while ((call&bit)==0);

        if (bit == 0x1) {
            devs_internal_transition();
        } else if (bit == 0x2) {
            devs_external_transition();
        } else {
            devs_confluence_transition();
        }
        call &= ~bit;
    }
}

if (extTransitionOccured && intTransitionOccured) {
    nextState = conState;
} else if (extTransitionOccured && !intTransitionOccured) {
    nextState = extState;
} else if (intTransitionOccured && !extTransitionOccured) {
    nextState = intState;
} else {
    cerr << ERROR() << std::setw(8) << "###ERROR_in_" << name() << ": devs_advance() called without a correct optimize level (no OPT_ALL_SERIAL, OPT_NEEDED_SERIAL, OPT_ALL_PARALLEL given) ###" << NORMAL() << endl << BACKTRACE() << endl;
BACKTRACE() << endl;
nextState = initialize(); // fallback. this should never be hit!

if (debugLevel&DBG_ADVANCE) {
    cout << ADVANCE() << std::setw(12) << timestamp("\":\") << "["
        << devsnname() << ",devs_advance_with_int=" <<
    intTransitionOccured << "\"and\" ext=" <<
    extTransitionOccured << "\"\"
        << print(cout);
    cout << "\"\"\";
    print(cout, nextState);
    cout << NORMAL() << endl;
}

// output data, iff an internal transition occurred
if (intTransitionOccured)
    output(curState);

// update current state, update last active timestamp and set
// the new timeout for the internal transition
curState = nextState;
devs_update_time();
devs_timeout();

void devs_timeout(void) {
    devs_time next_timeout = time_advance(curState);
    if (debugLevel&DBG_TIMEOUT) cout << TIMEOUT() << std::setw(12)
        << timestamp("\":\") << "[" << devsnname() << ",devs_timeout_
    -->\" << next_timeout << NORMAL() << endl;
    timeoutEvent.cancel();
    if (next_timeout != DEVS_TIME_INF) {
        timeoutEvent.notify(next_timeout.get_sc_time());
    }
}

devs_core::devs_time devs_elapsed_time(void) {
    return sc_time_stamp() - lastActive;
}

void devs_update_time(void) {
    lastActive = sc_time_stamp();
}

std::string timestamp(const std::string post="") {
    return devs_core::timestamp((debugLevel&DBG_DELTA)!=0, post);
protected:
mutable sc_spawn_options devs_advance_options; // used to save the sensitivity of the component for the input port(s)

void register_input(devs_in_if *in_if) const {
    devs_advance_options.set_sensitivity(&(in_if->default_event()));
    listInputs.push_back(in_if);
}

// Model-specific functions
protected:
virtual void print(std::ostream& os, const State& state) const = 0;
virtual void register_input_ports(void) const = 0;
virtual State initialize(void) const = 0;
virtual void output(const State& state) const = 0;
virtual devs_core::devs_time time_advance(const State& state) const = 0;
virtual State external_transition(const State& state, const devs_core::devs_time& elapsed) const = 0;
virtual State internal_transition(const State& state) const = 0;
virtual State confluence(const State& state, const devs_core::devs_time& elapsed) const = 0;

private:
pthread_mutex_t advanceConditionMutex, returnToAdvanceConditionMutex; // Mutex to synchronize the access to the conditions below.
pthread_cond_t advanceCondition, returnToAdvanceCondition; // Conditions between {int, ext, con}−Transition and devs_advance.
 pthread_t externalThread, internalThread, confluenceThread; // The threads for the parallel version.
unsigned int parallelExecutionStatus; // Current status of the parallel executed threads, i.e. which is currently running.
unsigned int optimizeLevel; // Flags to indicate that the parallel next state calls should be serial calls.
State curState; // the current state of the devs;
State extState; // the next state after the external transition
State intState; // the next state after the internal transition
State conState; // the next state after the confluence transition
// (both internal and external events fired up the component)
devs_core::devs_time lastActive; // hold the timestamp the last transition occured
sc_event timeoutEvent; // event for the timeout of a state (fires internal_transition)
mutable std::list<devs_core::devs_in_if*> listInputs; // this list saves a reference to all used (registered) input ports
unsigned int debugLevel; // debug output is appreciated with verbosity 0=none,1=essential,2=all
template <class State> std::ostream& operator << (std::ostream& os, DEVS_Component<State>& component) {
    component.print(os);
    return os;
};

template <class State> void* run_devs_external_transition_thread (void* param) {
    DEVS_Component<State>* model = (DEVS_Component<State>*)param;
    model->devs_external_transition_thread();
    return NULL;
}

template <class State> void* run_devs_internal_transition_thread (void* param) {
    DEVS_Component<State>* model = (DEVS_Component<State>*)param;
    model->devs_internal_transition_thread();
    return NULL;
}

template <class State> void* run_devs_confluence_transition_thread (void* param) {
    DEVS_Component<State>* model = (DEVS_Component<State>*)param;
    model->devs_confluence_transition_thread();
    return NULL;
}

template <class State> class DEVS_Component_Advance_Callback {
public:
    DEVS_Component_Advance_Callback(DEVS_Component<State>* target) {
        target_p = target;
    }
    inline void operator () () {
        target_p->devs_initialize();
        while (true) {
            wait();
            target_p->devs_advance();
        }
    }
protected:
    DEVS_Component<State>* target_p;
};

Appendix A Source Codes
A.2 GPT Example

Listing A.2 Generator component

```
1 STATE_SET(Phase);
2 STATE(Phase, WAIT);
3 STATE(Phase, GENERATE);

4 struct GeneratorState {
5   Phase phase;
6   unsigned int count;
7   devs_time period;
8 };

9 class Generator : public DEVS_Component<GeneratorState> {
10 public:
11   DEVS_OUT<unsigned int> iOutput;
12   DEVS_IN<unsigned int> iStart;
13   DEVS_IN<unsigned int> iStop;
14   devs_time initPeriod;

15     Generator(sc_module_name name, devs_time _initPeriod) :
16       DEVS_Component<GeneratorState>(name) {
17       initPeriod = _initPeriod;
18     };

19     virtual void register_input_ports(void) const {
20       register_input(&iStart);
21       register_input(&iStop);
22     }

23     GeneratorState initialize(void) const {
24       GeneratorState ret = {WAIT, 0, DEVS_TIME_INF};
25       return ret;
26     }

27     void output(const GeneratorState& state) const {
28       iOutput = state.count;
29     }

30     devs_time time_advance(const GeneratorState& state) const {
31       return state.period;
32     }

33     GeneratorState external_transition(const GeneratorState& state,
34       const devs_time& elapsed) const {
35       if (state.phase == WAIT && iStart == 1) {
36         return GeneratorState
37             {
38               Phase {GENERATE},
39               count = count + 1,
40               initPeriod = initPeriod
41             };
42     }
```

```c
GeneratorState next_state;
next_state.phase = GENERATE;
next_state.count = state.count;
next_state.period = initPeriod;
return next_state;
} else if (state.phase == GENERATE && iStop == 1) {
GeneratorState next_state;
next_state.phase = WAIT;
next_state.count = state.count;
next_state.period = DEVS_TIME_INF;
return next_state;
}
return state;
}
GeneratorState internal_transition(const GeneratorState& state) const {
GeneratorState next_state;

next_state = state;
next_state.count++;

return next_state;
}
GeneratorState confluence(const GeneratorState& state, const devs_time& elapsed) const {
return external_transition(internal_transition(state), elapsed);
}

Listing A.3 Process component

STATE_SET(Processor);
STATE(Processor, WAITFORJOB);
STATE(Processor, PROCESS);

struct ProcessorState {
Processor phase;
unsigned int count;
devs_time period;
};

class Processor : public DEVS_Component<ProcessorState> {
public:
DEVS_OUT<unsigned int> iOutput;
DEVS_IN<unsigned int> iInput;
devs_time initPeriod;
```
Processor(sc_module_name name, devs_time _initPeriod) :

  DEVS_Component<ProcessorState >(name) {
    initPeriod = _initPeriod;
  }

virtual void register_input_ports(void) const {
  register_input(&iInput);
}

ProcessorState initialize(void) const {
  ProcessorState ret = {WAITFORJOB, 0, DEVS_TIME_INF};
  return ret;
}

void output(const ProcessorState& state) const {
  iOutput = state.count;
}

devs_time time_advance(const ProcessorState& state) const {
  return state.period;
}

ProcessorState external_transition(const ProcessorState& state,
  const devs_time& elapsed) const {
  ProcessorState next_state = state;
  if (state.phase == WAITFORJOB) {
    next_state.phase = PROCESS;
    next_state.count = iInput;
    next_state.period = initPeriod;
    return next_state;
  }
  next_state.period -= elapsed;
  return next_state;
}

ProcessorState internal_transition(const ProcessorState& state)
  const {
  return initialize();
}

ProcessorState confluence(const ProcessorState& state, const
devs_time& elapsed) const {
  return external_transition(internal_transition(state), elapsed);
}

}
Listing A.4 Observer component

```c++
STATE_SET(Observer);
STATE(Observer, OBSERVE);
STATE(Observer, END);

struct ObserverState {
    Observer phase;
    unsigned int in, out;
    devs_time period;
};

class Observer : public DEVS_Component<ObserverState> {
    public:
    DEVS_IN<unsigned int> iJobStart;
    DEVS_IN<unsigned int> iJobStop;
    DEVS_OUT<unsigned int> iStopGenerator;
    devs_time initPeriod;

    Observer(sc_module_name name, devs_time _initPeriod) :
        DEVS_Component<ObserverState>(name) {
        initPeriod = _initPeriod;
    }

    virtual void register_input_ports(void) const {
        register_input(&iJobStart);
        register_input(&iJobStop);
    }

    ObserverState initialize(void) const {
        ObserverState ret = {OBSERVE, 0, 0, initPeriod};
        return ret;
    }

    void output(const ObserverState& state) const {
        iStopGenerator = 1;
    }

    devs_time time_advance(const ObserverState& state) const {
        return state.period;
    }

    ObserverState external_transition(const ObserverState& state,
                                           const devs_time& elapsed) const {
        ObserverState next_state = state;
        if (state.phase == OBSERVE) {
            if (iJobStart.event())
                next_state.in++;
            if (iJobStop.event())
```
Listing A.5 `sc_main` function implementing the `gpt` example

```cpp
int sc_main(int argc, char* argv[]) {
    devs_signal<unsigned int> gen_out;
    devs_signal<unsigned int> proc_obs;
    devs_signal<unsigned int> obs_gen;
    devs_signal<unsigned int> tb_out;
    unsigned int g=1, p=1, t=10000;

    Generator generator("Generator", devs_time(g, SC_NS));
generator.optimize(OPT_NEEDED_SERIAL);
generator.iStart(tb_out);
generator.iStop(obs_gen);
generator.iOutput(gen_out);

    Processor processor("Processor", devs_time(p, SC_NS));
processor.optimize(OPT_NEEDED_SERIAL);
processor.iInput(gen_out);
processor.iOutput(proc_obs);

    Observer observer("Observer", devs_time(t, SC_NS));
observer.optimize(OPT_NEEDED_SERIAL);
observer.iJobStart(gen_out);
observer.iJobStop(proc_obs);
observer.iStopGenerator(obs_gen);

    // initialize model and run it
    generator.devs_startup();
}```
processor.dev_startup();
observer.dev_startup();
sc_start();
return (0);
A.3 UART Receiver (VHDL)

Listing A.6 UART Receiver VHDL source code automatically generated from the SynDEVS model

```
library IEEE;
use IEEE.NUMERIC_BIT.ALL;
library devs_uart_v1_00_a;
use devs_uart_v1_00_a.FUNCTIONS.ALL;

entity RECEIVER is
  generic (
    timer_width : integer := 10;
    default_value : integer := 0
  );
  port ( 
    clk: in bit;
    reset: in bit;
    rx: in bit;
    rx_enable: in bit;
    data_received: out unsigned(7 downto 0);
    data_received_enable: out bit;
    data_error: out bit;
    data_error_enable: out bit;
    timer_stop_in: in bit;
    disable_in: in bit
  );
end entity;

architecture BEHAVIOURAL of RECEIVER is
  type states_type is (state_idle, state_startbit, 
                     state_waitforfirstbit, state_readbit, state_parity, 
                     state_waitstopbit);
  signal state: states_type := state_idle;
  signal data: unsigned(7 downto 0) := TO_UNSIGNED(0,8);
  signal bitsread: unsigned(3 downto 0) := TO_UNSIGNED(0,4);
  signal evenparity: bit := '0';
  signal stop: bit := '0';
  signal timer: unsigned(timer_width - 1 downto 0) := TO_UNSIGNED( 
    default_value , timer_width);
begin
  RECEIVER: process (clk, reset) is
    begin
      if (reset='0') then
        if (clk'event and clk='1') then
          if (timer_stop_in='0') then
            timer <= devs_uart_v1_00_a.FUNCTIONS.decrement(timer);
```
end if;
if (timer=TO_UNSIGNED(0,timer_width)) then
  timer <= TO_UNSIGNED(default_value,timer_width);
end if;
if (disable_in='1') then
  data_received_enable <= '0';
data_error_enable <= '0';
end if;
case (state) is
when state_idle =>
  if (disable_in='1' and rx='0' and rx_enable='1') then
    timer <= TO_UNSIGNED(433,10);
    state <= state_startbit;
  end if;
when state_startbit =>
  if (TIMER /= TO_UNSIGNED(0,timer_width) and disable_in
      = '1' and rx='1' and rx_enable='1') then
    state <= state_idle;
  end if;
  if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
      and rx='0' and rx-enable='1') then
    timer <= TO_UNSIGNED(867,10);
data <= TO_UNSIGNED(0,8);
bitsread <= TO_UNSIGNED(0,4);
evenparity <= '0';
stop <= '0';
state <= state_waitforfirstbit;
end if;
when state_waitforfirstbit =>
  if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
      and rx-enable='1') then
    timer <= TO_UNSIGNED(867,10);
data <= rx&data(7 downto 1);
bitsread <= bitsread+TO_UNSIGNED(1,4);
evenparity <= rx xor evenparity;
state <= state_readbit;
end if;
when state_readbit =>
  if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
      and rx-enable='1' and (not (bitsread=TO_UNSIGNED
          (8,4)))) then
    timer <= TO_UNSIGNED(867,10);
data <= rx&data(7 downto 1);
bitsread <= bitsread+TO_UNSIGNED(1,4);
evenparity <= rx xor evenparity;
state <= state_readbit;
end if;
if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
    and rx-enable='1' and bitsread=TO_UNSIGNED(8,4))
then
  timer <= TO_UNSIGNED(867,10);
  evenparity <= rx xor evenparity;
  state <= state_parity;
end if;
when state_parity =>
  if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
      and rx_enable='1') then
    timer <= TO_UNSIGNED(433,10);
    stop <= rx;
    state <= state_waitstopbit;
  end if;
begin
  when state_waitstopbit =>
  if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
      and rx_enable='1' and (stop='1' and evenparity='0'))
    then
    data_received <= data;
    data_received_enable <= '1';
    state <= state_idle;
  end if;
  if (disable_in='1' and TIMER=TO_UNSIGNED(0,timer_width)
      and rx_enable='1' and (not (stop='1' and evenparity
                               = '0'))) then
    data_error <= '1';
    data_error_enable <= '1';
    state <= state_idle;
  end if;
  end case;
end if;
else
  timer <= TO_UNSIGNED(default_value,timer_width);
  state <= state_idle;
  data <= TO_UNSIGNED(0,8);
  bitsread <= TO_UNSIGNED(0,4);
  evenparity <= '0';
  stop <= '0';
  data_received <= (others => '0');
  data_received_enable <= '0';
  data_error <= '0';
  data_error_enable <= '0';
end if;
end process RECEIVER;
end architecture BEHAVIOURAL;
A.4 UART Transceiver (C++)

Listing A.7 Atomic_Component_transmitter class declaration with additional declaration of the component’s states

```cpp
1 #ifndef ATOMIC_COMPONENT_TRANSMITTER_H_
2 #define ATOMIC_COMPONENT_TRANSMITTER_H_
3
4 #include "ComponentState.h"
5 #include "Atomic_Component.h"
6 #include "RegisterFile.h"
7 #include "Port.h"
8 #include "model_definitions.h"
9 #ifdef DEBUG
10 #include "xparameters.h"
11 #include "stdio.h"
12 #endif /* DEBUG */
13
14 extern RegisterFile registerFile;
15
16 // State IDS
17 #define STATE_transmitter_IDLE 1
18 #define STATE_transmitter_READDATA 2
19 #define STATE_transmitter_SENDBIT 3
20 #define STATE_transmitter_SENDPARITY 4
21 #define STATE_transmitter_INIT 5
22 #define STATE_transmitter_STOPBIT 6
23 #define STATE_transmitter_STARTBIT 7
24
25 class ComponentState_transmitter_idle : public ComponentState {
26 public:
27     ComponentState_transmitter_idle() _CONST_INIT_SECTION:
28     ComponentState(STATE_transmitter_IDLE , TIME_INFINITE) {};
29 }
30
31 class ComponentState_transmitter_readdata : public ComponentState {
32 public:
33     ComponentState_transmitter_readdata() _CONST_INIT_SECTION:
34     ComponentState(STATE_transmitter_READDATA , 0) {};
35 }
36
37 class ComponentState_transmitter_sendbit : public ComponentState {
38 public:
39     ComponentState_transmitter_sendbit() _CONST_INIT_SECTION:
40     ComponentState(STATE_transmitter_SENDBIT , 0) {};
41 }
```
class ComponentState_transmitter_sendparity : public ComponentState{
    
public:
    ComponentState_transmitter_sendparity () _CONST_INIT_SECTION:
        ComponentState(ST ATE_transmitter_SENDPARITY ,0) {};
};

class ComponentState_transmitter_init : public ComponentState {
    
public:
    ComponentState_transmitter_init () _CONST_INIT_SECTION:
        ComponentState(ST ATE_transmitter_INIT ,0) {};
};

class ComponentState_transmitter_stopbit : public ComponentState {
    
public:
    ComponentState_transmitter_stopbit () _CONST_INIT_SECTION:
        ComponentState(ST ATE_transmitter_STOPBIT ,0) {};
};

class ComponentState_transmitter_startbit : public ComponentState {
    
public:
    ComponentState_transmitter_startbit () _CONST_INIT_SECTION:
        ComponentState(ST ATE_transmitter_STARTBIT ,0) {};
};

class Atomic_Component_transmitter : public Atomic_Component {
    
public:
    Atomic_Component_transmitter () _CONST_INI T_SECTION :
        Atomic_Component(&registerFile . cu rrentState_transmitter ) ,
        variable_send(&registerFile . variable_transmitter_send ) ,
        variable_parity(&registerFile . variable_transmitter_parity ) ,
        variable_bits(&registerFile . variable_transmitter_bits ) ,
        variable_transmit(&registerFile . variable_transmitter_transmit )
    { };

void reset () const; // override
void eventRun () const; // override

// Output Ports
static Port1Bit* const outport_dataack ;
static Port1Bit* const outport_tx ;

// Input Ports
static PortU8Bit* const inport_data ;

// Variables
int32_t* const variable_send ;
int32_t* const variable_parity ;
int32_t* const variable_bits ;
int32_t* const variable_transmit ;

private:

    // States
ComponentState_transmitter_idle const state_idle ;
Listing A.8 Atomic_Component_transmitter class definition C++ source code for an UART transceiver

```cpp
#include "Atomic_Component_transmitter.h"

void Atomic_Component_transmitter::reset() const {
    // Output init
    this->outport_dataack->data = 0;
    this->outport_dataack->isActive = false;
    this->outport_tx->data = 0;
    this->outport_tx->isActive = false;
    // Variable initialization
    *variable_send = 0;
    *variable_parity = 0;
    *variable_bits = 0;
    *variable_transmit = 0;
    // Set Initial State
    startNewState((ComponentState * const) &state_init);
}

void Atomic_Component_transmitter::eventRun() const {
    this->outport_dataack->isActive = false;
    this->outport_tx->isActive = false;
    switch (this->current_state->current_state_id) {
    case STATE_transmitter_IDLE:{ // state_idle
        if (this->inport_data->isActive) {
            register int tempVariable_send = (this->inport_data->data);
            register int tempVariable_parity = 0;
            register int tempVariable_bits = 0;
            *variable_send = tempVariable_send;
            *variable_parity = tempVariable_parity;
            *variable_bits = tempVariable_bits;
            *variable_transmit = tempVariable_transmit;
            startNewState((ComponentState * const) &state_readdata);
        }
        break;
    }
```

case STATE_transmitter_READDATA:
    //=state_readdata
    if (this->current_state->current_time == 0) {
        this->outport_dataack->data = 1;
        this->outport_dataack->isActive = true;
        this->outport_tx->data = (*variable_transmit);
        this->outport_tx->isActive = true;
        register int tempVariable_send = (0<<7 | (*variable_send>>1 & 127));
        register int tempVariable_parity = (*variable_send>>0 & 1);
        register int tempVariable_transmit = (*variable_send>>0 & 1);
        *variable_send = tempVariable_send;
        *variable_parity = tempVariable_parity;
        *variable_transmit = tempVariable_transmit;
        startNewState((ComponentState * const) &state_startbit);
    }
    break;
}

case STATE_transmitter_SENDBIT:
    //=state_sendbit
    if (this->current_state->current_time == 0 & & !this->inport_data->isActive & & (!(((*variable_bits)==8)) )) {
        this->outport_tx->data = (*variable_transmit);
        this->outport_tx->isActive = true;
        register int tempVariable_send = (0<<7 | (*variable_send>>1 & 127));
        register int tempVariable_parity = (*variable_send>>0 & 1);
        register int tempVariable_bits = ((*variable_bits) + 1);
        register int tempVariable_transmit = (*variable_send>>0 & 1);
        *variable_send = tempVariable_send;
        *variable_parity = tempVariable_parity;
        *variable_bits = tempVariable_bits;
        *variable_transmit = tempVariable_transmit;
        startNewState((ComponentState * const) &state_sendbit);
    } else
    if (this->current_state->current_time == 0 & & !this->inport_data->isActive & & ( (*variable_bits)==8 ) ) {
        this->outport_tx->data = (*variable_parity);
        this->outport_tx->isActive = true;
        *variable_transmit = 1;
        startNewState((ComponentState * const) &state_sendparity);
    } else
    if (this->current_state->current_time == 0 & & !(((*variable_send)==8)) ) {
        this->outport_tx->data = (*variable_transmit);
        this->outport_tx->isActive = true;
        register int tempVariable_send = (0<<7 | (*variable_send>>1 &

Appendix A Source Codes

79 register int tempVariable_parity = (*variable_parity) ^ (*variable_send>>0 & 1);
80 register int tempVariable_bits = ((*variable_bits) + 1);
81 register int tempVariable_transmit = (*variable_send>>0 & 1);
82 *variable_send = tempVariable_send;
83 *variable_parity = tempVariable_parity;
84 *variable_bits = tempVariable_bits;
85 *variable_transmit = tempVariable_transmit;
86 startNewState((ComponentState * const)&state_sendbit);
87 } else
88 if (this->current_state->current_time == 0 && (*variable_bits) == 8) {
89 this->outport_tx->data = (*variable_parity);
90 this->outport_tx->isActive = true;
91 *variable_transmit = 1;
92 startNewState((ComponentState * const)&state_sendparity);
93 }
94 break;
95 }
96 case STATE_transmitter_SENDPARITY:{ //=state_sendparity
97 if (this->current_state->current_time == 0) {
98 this->outport_tx->data = (*variable_transmit);
99 this->outport_tx->isActive = true;
100 startNewState((ComponentState * const)&state_stopbit);
101 }
102 break;
103 }
104 }
105 case STATE_transmitter_INIT:{ //=state_init
106 if (this->current_state->current_time == 0) {
107 this->outport_tx->data = 1;
108 this->outport_tx->isActive = true;
109 startNewState((ComponentState * const)&state_idle);
110 }
111 break;
112 }
113 }
114 case STATE_transmitter_STOPBIT:{ //=state_stopbit
115 if (this->current_state->current_time == 0) {
116 startNewState((ComponentState * const)&state_idle);
117 }
118 break;
119 }
120 }
121 case STATE_transmitter_STARTBIT:{ //=state_startbit
122 if (this->current_state->current_time == 0) {
123 this->outport_tx->data = (*variable_transmit);
A.4 UART Transceiver (C++)

```c
125     this->outport_tx->isActive = true;
126     register int tempVariable_send = (0<<7 | (*variable_send>>1 & 127));
127     register int tempVariable_parity = (*variable_parity) ^ (*variable_send>>0 & 1);
128     register int tempVariable_bits = ((*variable_bits) + 1);
129     register int tempVariable_transmit = (*variable_send>>0 & 1);
130     *variable_send = tempVariable_send;
131     *variable_parity = tempVariable_parity;
132     *variable_bits = tempVariable_bits;
133     *variable_transmit = tempVariable_transmit;
134     startNewState((ComponentState * const) &state_sendbit);
135 }
136    break;
137 }
138 }
139 #ifdef DEBUG
140   xil_printf("transmitter:STATE=%d,%d\r\n", (int) this->
141         current_state->current_state_id, (int) this->current_state->
142         current_time);
143   if (this->inport_data->isActive)
144     xil_printf("In-Event_‘data’=0x%X\r\n", (int) this->inport_data->
145         data);
146   if (this->outport_dataack->isActive)
147     xil_printf("Out-Event_‘dataack’=0x%X\r\n", (int) this->
148         outport_dataack->data);
149   if (this->outport_tx->isActive)
150     xil_printf("Out-Event_‘tx’=0x%X\r\n", (int) this->outport_tx->
151         data);
152 #endif /* DEBUG */
153 }
APPENDIX B

List of Publications


8. H. Gregor Molter, André Seffrin, and Sorin A. Huss. DEVS2VHDL: Automatic Transformation of XML-specified DEVS Model of Computation into Synthesiz-


APPENDIX C

List of Supervised Theses

1. Nico Huber. Quantenresistente Kryptograhieverfahren: Entwurf und Implemen
tierung einer Hardware-Architektur des Merkle-Signaturverfahrens, Februar

Diploma Thesis.


4. Thorsten Wink. Quantenresistente Kryptograhieverfahren, January 2009. Bach-
elor Thesis.

und flexible Multi-Core-Architektur als System-on-a-Chip inklusive OpenSSL

Diploma Thesis.

7. Nabil Sayegh. Sichere Kommunikation in eingebetteten Systemen am Beispiel

8. Johannes Kohlmann. Generierung eingebetteter Systeme aus DEVS-Modellen,

References


References


[HOS+07] Andreas Herrholz, Frank Oppenheimer, Andreas Schallenberg, Wolfgang Nebel, Christoph Grimm, Markus Damm, Fernando Herrera,


References


Mappings. In 4th IFAC Workshop on Discrete-Event System Design (DES-Des’09), October 2009.


References


References


