

Synthesis of Software Programs

Presented by
Betül Büyükkurt

Embedded System Design Trends

- ◆ Embedded systems implementations vary from full hardware to full software.
- ◆ Full hardware implementations:
 - tasks are translated into a set of customized integrated circuits.
- ◆ Full software implementations:
 - the system is implemented as software routines running on standard components like MPUs or DSPs
- ◆ There is a shift towards mixed designs with more software component each day.
 - Software flexibility
 - Increased time-to-market pressure

Compilation vs Synthesis

- ◆ Goal: "right-for-the-first-time" designs.
- ◆ Software Compilation:
 - Non-concurrent input languages. E.g. C, Pascal
 - Input language syntax & semantics are very close to that of the implementation
 - Source codes describe the implementation
- ◆ Software Synthesis:
 - Aggressive optimization
 - Higher level functional specification, rather than how the functionality is implemented

Software vs Hardware Compilation

- | | |
|--|--------------------------------------|
| ◆ Software Compilation: | ◆ Hardware Compilation: |
| ▪ Register allocation | ▪ Functional and register allocation |
| ▪ Instruction selection | ▪ Scheduling |
| ▪ Local optimizations | ▪ Component synthesis |
| ▪ Global optimization = long compilation times | ▪ Optimization |

Software Synthesis

- ◆ Product of system level design approach
 - Hardware and software components of a system are synthesized together
- ◆ Software components are derived from the same starting point as hardware
 - It is like:
 - Small set of simple constructs
 - No pointers, recursion, runtime determined loop bounds
- ◆ Aggressive global optimization techniques are applied as in hardware

Environment

- ◆ Goal: Decreasing the development time for embedded systems
- ◆ Method:
 - Select an implementation independent high-level representation
 - Easy to use for the system designer
 - Translate to lower levels using a set of mathematically well-defined operations
 - Optimize the new representation
- ◆ Focus of the studied paper
 - System choice: Control dominated embedded applications
 - Representation choice: Interconnection network of communication processes with FSM semantics

CodesignFSMs

- ♦ FSM
 - Easily understood
 - Widely used as specifications
 - Abundant theoretical and practical results concerning their implementation
- ♦ CFSMs
 - Extends classical FSM with arithmetic and logical operators
 - Interacts via an asynchronous communication mechanism
 - Advantages: Flexibility and better expressive power

Software Synthesis Approach

- ♦ Initial specification: Network of CFSMs
 - Network: asynchronous, concurrent, and dynamically scheduled
 - Individual CFSMs: synchronous, statically scheduled
 - Granularity of the individual CFSMs can be supplied as a parameter
- ♦ Translate CFSMs into C code
 - The produced code is simple and low-level
 - Easy timing analysis
 - Allows a very tight control over machine code
- ♦ Intermediate Data Structure:
 - s-graph: a simple control/data flow graph, stands for "software"
 - Easy to translate into C
 - Has only conditional branch and assignment as primitives

Software Synthesis Procedure

- ♦ Perform an optimized translation of CFSM into s-graph
 - Builds an optimized binary decision diagram (BDD) as an intermediate representation for the translation process into s-graphs
- ♦ Optimizes s-graph and estimates code size
- ♦ Translates s-graph into C
- ♦ Schedule CFSMs and generate the RTOS
- ♦ Compile into machine code (general purpose C compiler)

Binary Decision Diagrams

- ♦ It is the key data structure that is used as an intermediate representation between CFSMs and C
- ♦ Provides efficient representation for storing and manipulating Boolean functions
- ♦ BDD is a directed acyclic graph with a root node for each output function and leaf nodes representing the value of each output function for each input minterm.
 - Each non-leaf node is associated with an input variable
 - Each non-leaf node has two outgoing edges, each representing the value of the input variable (zero or one) along that branch
- ♦ Given a function and an ordering of the input variables, the reduced-ordered BDD is a canonical form of it

Characteristic Functions

characteristic functions. A single-output binary-valued function $\chi^f : (X \times Y) \rightarrow \{0,1\}$, where $X = X_1 \times \dots \times X_m$ and $Y = Y_1 \times \dots \times Y_l$, represents the multioutput multivalued function $f : X \rightarrow Y$ if $\chi^f(x,y) \Leftrightarrow (y = f(x))$. The same notation can also be used to describe a relation R , as $\chi^R(x,y) \Leftrightarrow (y \in R(x))$.

if $x_j \in \{0,1\}$, then $S_{x_j} f = f_{x_j=1} \vee f_{x_j=0}$.
 The support of an output variable y_i of a multioutput function is the set of inputs upon which y_i essentially depends. More precisely, an input variable x_j belongs to the support of y_i if $S_{x_j} y_i(x_1, \dots, x_n) \neq y_i(x_1, \dots, x_n)$.

Networks of CodesignFSMs

- ♦ Operates globally as asynchronous, locally synchronous
- ♦ Each CFSM
 - detects atomically as a snapshot of its input events, and performs site calculations independently and asynchronously of other CFSMs
 - As a result of the computation, a CFSM may later change its state and/or emit output events

Communication of CFSMs

- ◆ CFSMs communicate via events
 - Each event is associated with a boolean flag, indicating the presence of the event
 - A flag is set by the emitter and can either be at the same time or some small time before the event value gets updated.
- ◆ Each input event may be detected at most once after its emission
- ◆ There is no guarantee that an event will be detected before it is overwritten again
 - Assumption is that there is a buffer of length one between each CFSM and for each event.

Software Graphs

- ◆ An s-graph is a directed acyclic graph with one source and one sink
- ◆ Has four types of vertices: BEGIN, END, TEST, ASSIGN
- ◆ Source → BEGIN, sink → END
- ◆ Each TEST vertex has two children: true(v), false(v);
- ◆ Each ASSIGN or BEGIN vertex has one child and i.e. next(u)

Software Graphs – Cont...

- ◆ An s-graph is associated with a set of m input and l output variables: z_1, \dots, z_{m+l}
 - ASSIGN: $z_v := a_v(z_1, \dots, z_{m+l})$
 - TEST: $p_v(z_1, \dots, z_{m+l})$
- ◆ To each valued input, two inputs - graph variables are defined; one boolean and one with the same domain as the signal

S-Graph Example

```

module simple:
input c: integer;
output y;
var a: integer in
loop
await c;
if a = ?c then
a := 0;
emit y;
else
a := a + 1;
end if
end loop
end var
end module
  
```

```

procedure build( $z_1, \dots, z_{m+l}$ ; variable;  $\bar{v}$ : index;  $F$ : function)
begin
if  $\bar{v} = 0$  then
create a BEGIN vertex  $v$ 
next( $v$ )  $\leftarrow$  build( $z_1, \dots, z_{m+l}, \bar{v} + 1, F$ );
else if  $F = 1$  then
create the END vertex  $v$ ;
else if  $z_{\bar{v}}$  is an input, then
create a TEST vertex  $v$  with  $p_v = (z_{\bar{v}})$ ;
true( $v$ )  $\leftarrow$  build( $z_1, \dots, z_{m+l}, \bar{v} + 1, F_{z_{\bar{v}}=1}$ )
false( $v$ )  $\leftarrow$  build( $z_1, \dots, z_{m+l}, \bar{v} + 1, F_{z_{\bar{v}}=0}$ )
else if  $z_{\bar{v}}$  is an output, then
create an ASSIGN vertex  $v$  labeled with  $z_{\bar{v}}$ 
and  $a_v = S_{z_{\bar{v}}}$ 
next( $v$ )  $\leftarrow$  build( $z_1, \dots, z_{m+l}, \bar{v} + 1, S_{z_{\bar{v}}}, F$ )
return reduce( $v$ )
end
  
```

```

procedure evaluate ( $v$ : vertex;  $x_1, \dots, x_m$ : variable)
begin
for  $1 \leq j \leq m + l$ 
if  $z_j$  is an input then assign to it the corresponding  $x_j$ 
else  $z_j \leftarrow \epsilon$ 
evalStep (next( $v$ ),  $x_1, \dots, x_{m+l}$ )
for  $1 \leq j \leq m + l$ 
if  $z_j$  is an output then assign it to the corresponding  $y_j$ 
return ( $y_1, \dots, y_l$ )
end

procedure evalStep ( $v$ : vertex;  $x_1, \dots, x_{m+l}$ : variable)
begin
if  $v$  is a TEST then
if  $p_v(x_1, \dots, x_{m+l})$  then
evalStep (true( $v$ ),  $x_1, \dots, x_{m+l}$ )
else evalStep (false( $v$ ),  $x_1, \dots, x_{m+l}$ )
else if  $v$  is an ASSIGN then
 $z_v \leftarrow a_v(x_1, \dots, x_{m+l})$ 
evalStep (next( $v$ ),  $x_1, \dots, x_{m+l}$ )
end
  
```

S-Graph Optimization

- ♦ By Reordering
- ♦ Ordering Outputs after their support
- ♦ Ordering outputs before their support
- ♦ Optimization by collapsing test nodes

S-Graph to C translation

- ♦ Straightforward
- ♦ After initial variable declarations with appropriate types and initializations:
 - A TEST node is translated into if or switch statements with appropriate number of goto constructs
 - An ASSIGN node is translated into an assignment statement.

Cost and Performance Estimation

- ♦ Each vertex on the S-graph is assigned two cost parameters, one for timing and one for size.
- ♦ Edges may also have associated costs, e.g. the cost parameters of a TEST node for each edge are stored explicitly.
- ♦ Different kinds of statements are associated with different costs:
 - A TEST node detecting the presence of a signal
 - A TEST node branching on a multivalued expression
 - An ASSIGN node emitting a signal
 - An ASSIGN node which assigns an expression to a variable

Cost and Performance Estimation – Cont...

- ♦ Other parameters:
 - Initialization of local variable
 - Execution times for predefined sw libraries
 - Calling and returning from C functions given local variable and parameters
 - Cost of goto() operations
 - Costs for different variables sizes
 - ...
- ♦ Cost parameters are determined for each target system.
- ♦ The value of each parameter is determined by examining the execution cycles.

Generating the RTOS

- ♦ Schedule individual swCFSMs such that each satisfies its timing constraints
 - Scheduling policy can be a user defined scheduling mechanism
- ♦ Provide mechanism for event emission and detection between individual swCFSMs
 - Events of each CFMS are associated with flags
 - A CFMS is ready to be scheduled for run by the RTOS whenever at least one input flag is set
- ♦ Provide mechanism for transferring the events between CFMSs that are implemented in hardware and that are implemented in software
 - Polling/Interrupts
- ♦ Ensure the consumption of events is consistent with the system semantics