Model Based Development of Embedded Control Software

Part 5: Portable TDL Run-time System

Claudiu Farcas

Credits: MoDECS Project Team, Giotto
Department of Computer Science
cs.uni-salzburg.at

UNIVERSITÄT SALZBURG

Contents

- Compiling TDL modules
- Run-Time support
- Platform specifics

Generic implementation details

- Plugin for TDL Compiler for generating C code
- TDL Run-Time System (E-machine*)
  - Reactivity code – Ecode & Drivers
  - Schedulability code – User defined scheduler with support for distribution
  - Support for safe system startup/shutdown
- TDL-Comm subsystem
  - Access to Public TDL ports of the module
  - Synchronized comunication via Real-Time Ethernet or IEEE 1394a/b (Firewire) or CAN/TTCAN

Plugin for TDL Compiler

- ANSI-C Platform Abstraction Layer
  - Static TDL Port Allocation
    - Basic types
    - Opaque types
  - Implicit Driver calls
  - Guards
  - Task Launchers
    - Used to pass parameters to tasks
    - Return control to TDL-RTS for improved Idle time utilization
  - E-Code encapsulation

Sample TDL Module – M1

```
module M1 {
  public const
    c1 = 50; c2 = 200;
    refPeriod = 10ms;

  sensor
    int s uses getS;

  actuator
    int a1 := c1 uses setA1;
    int a2 := c2 uses setA2;

  public task inc [wcet=1ms] {
    output int o := c1;
    uses incImpl(o);
  }

  public task dec [wcet=1ms] {
    output int o := c2;
    uses decImpl(o);
  }
```

```
start mode m1 [period=refPeriod] {
    task
      [1] inc();
      [1] dec();
    actuator
      [1] a1 := inc.o;
      [1] a2 := dec.o;
    mode
      [1] if switch2m2(s, inc.o) then m2;
  }

  mode m2 [period=refPeriod] {
    task
      [1] inc();
      [2] dec();
    actuator
      [1] a1 := inc.o;
      [2] a2 := dec.o;
    mode
      [1] if switch2m1(s, inc.o) then m1;
  }
} // End module
```


UNIVERSITÄT SALZBURG

Compiled TDL Ports, Guards

```
// Ports definitions
TDL_Int _TDL_M1_Port0 = 50; // M1_a1
TDL_Int _TDL_M1_Port1 = 200; // M1_a2
TDL_Int _TDL_M1_Port2; // M1_s
TDL_Int _TDL_M1_Port3 = 200; // M1_dec_o public
TDL_Int _TDL_M1_Port3_VAL = 200; // actual value of output o0
TDL_Int _TDL_M1_Port4 = 50; // M1_inc_o public
TDL_Int _TDL_M1_Port4_VAL = 50; // actual value of output o1

// Guards (one possible implementation without function pointers)
TDL_Boolean _TDL_GUARDS_M1(TDL_Int n) // Boolean type as char/byte
{
   switch (n)
   {
        case 0: return M1_switch2m2(_TDL_M1_Port2, _TDL_M1_Port4);
        case 1: return M1_switch2m1(_TDL_M1_Port2, _TDL_M1_Port4);
        default: _TDL_Throw_Exception(_TDL_INVALID_GUARD_NUMBER);
   }
   return(0);
}
```


UNIVERSITÄT SALZBURG

Implicit generation of Driver calls

```
//Drivers – Encapsulation of reactive behavior (zero logical time)
void _TDL_DRIVERS_M1(TDL_Int n)
{
    switch (n)    //hopefully the compiler optimizes this, otherwise use separate
    {             //functions and use a lookup mechanism
          case 0: //start task dec
                   _TDL_Release_Task(TDL_Launcher_M1_dec, 0, _TDL_MODULEID_M1);
                   break;
    [...] case 2: //terminate task dec
                   _TDL_M1_Port3 = _TDL_M1_Port3_VAL; // copy on termination
                   break;
    [...] case 5: //terminate task inc
                   _TDL_M1_Port4 = _TDL_M1_Port4_VAL; // copy on termination
                   break;
          case 6: //set M1_a1
                   M1_setA1(_TDL_M1_Port0);
                   break;
    [...] case 10: //actuator update a1
                   _TDL_M1_Port0 = _TDL_M1_Port4;
                   break;
    [...] case 12: //get M1_s
                   _TDL_M1_Port2 = M1_getS();
                   break;
    [...]
          default: _TDL_Throw_Exception(_TDL_INVALID_DRIVER_NUMBER);
    }
}
```

Platform Abstraction Layer for Tasks

```
TASK(TDL_Launcher_M1_dec) // Wrapper for the actual M1_dec functionality
{
_TDL_THREADING_START(0, _TDL_MODULEID_M1); // Macro PAL expansion
        M1_decImpl(&_TDL_M1_Port3_VAL);
_TDL_THREADING_END; // Signal completion to E-Machine
}

TASK(TDL_Launcher_M1_inc) // Wrapper for the actual M1_inc functionality
{
_TDL_THREADING_START(1, _TDL_MODULEID_M1); // Macro PAL expansion
        M1_incImpl(&_TDL_M1_Port4_VAL);
_TDL_THREADING_END; // Signal completion to E-Machine
}
```

POSIX/InTime PAL Mapping:

_TDL_THREADING_START(x,y) => while(1) { _TDL_WaitSignal(x,y);

OSEK PAL Mapping: no real functionality

UNIVERSITÄT SALZBURG

Adding timing specifications

```
// Static E-code embedding into C source file via macros
_TDL_ECode _TDL_ECODES_M1[_TDL_ECODES_COUNT_M1] = {
// initialization
    CALL(6),                 // #0 actuator init: setA1(a1)
    CALL(7),                 // #1 actuator init: setA2(a2)
    RETURN(),                // #2 return

// Start Mode: m1[0] starting at: #3 period: 10000
    CALL(8),                 // #3 call 8 task input update: inc
    RELEASE(1,3,10000),      // #4 release 1 : incImpl [1] until  10000 wcet = 1000
    CALL(9),                 // #5 call 9 task input update: dec
    RELEASE(0,0,10000),      // #6 release 0 : decImpl [0] until  10000 wcet = 1000
    FUTURE(0,9,10000),       // #7 future jump to #9 after delta: 10000
    RETURN(),                // #8 return
    CALL(5),                 // #9 terminate task: inc
    CALL(2),                 // #10 terminate task: dec
    CALL(10),                // #11 call 10 actuator update: a1 := o
    CALL(6),                 // #12 call 6 actuator setter: setA1(a1)
    CALL(11),                // #13 call 11 actuator update: a2 := o
    CALL(7),                 // #14 call 7 actuator setter: setA2(a2)
    CALL(12),                // #15 call 12 get: s := getS()
    IF(0,17,19),             // #16 if switch2m2 (guard 0) then jump #17 else jump #19
    CALL(13),                // #17 call 13 mode switch driver
    SWITCH(1),               // #18 switch to mode 1 mode switch -> m2:0 at: 20
    JUMP(3),                 // #19 jump 3 next cycle: m1
[...]
Possible macros: CALL(driver_nr) -> OPCODE_CALL, driver_nr, -1, -1
    RELEASE(task_nr,driver_nr,deadline) -> OPCODE_RELEASE, task_nr, driver_nr, deadline
```


UNIVERSITÄT SALZBURG

TDL Run-time Support

- Clear separation of reactivity and schedulability code
- Fast execution of E-code with traps for illegal instructions and timeouts
- Support for user defined schedule
- Universal code base – compile to “any” machine
  - Platform specific construction embedded via macros
  - ANSI-C Compliant code
- Lightweight in both CPU and memory resources
  (x86 + debug = 4KB, MPC555 = 11KB)
- Support for distribution - networking ready API

InTime/POSIX Specifics

- Application = real-time process
  - TDL-RTS = high priority main thread
  - TDL Tasks = low priority user level threads
- Release Task = new thread with default priority (low)
- Terminate Task = terminate user function and return to E-Machine Scheduler
- Timing = kernel level Alarm event
  - max frequency 10kHz (minimum interval 100us)
  - exception handling for overruns
- User level scheduling via dynamic thread priority assignment

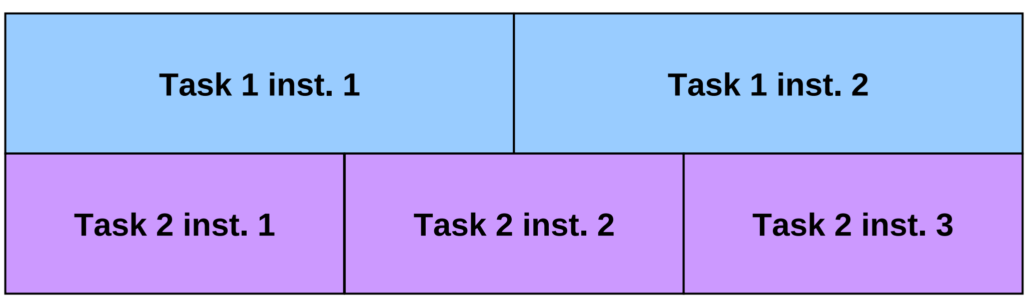
UNIVERSITÄT SALZBURG

InTime/POSIX Scheduling

- E-Machine selects next running task by raising/lowering thread priorities
- Released tasks
  - Low prio - sleep
  - High prio - run

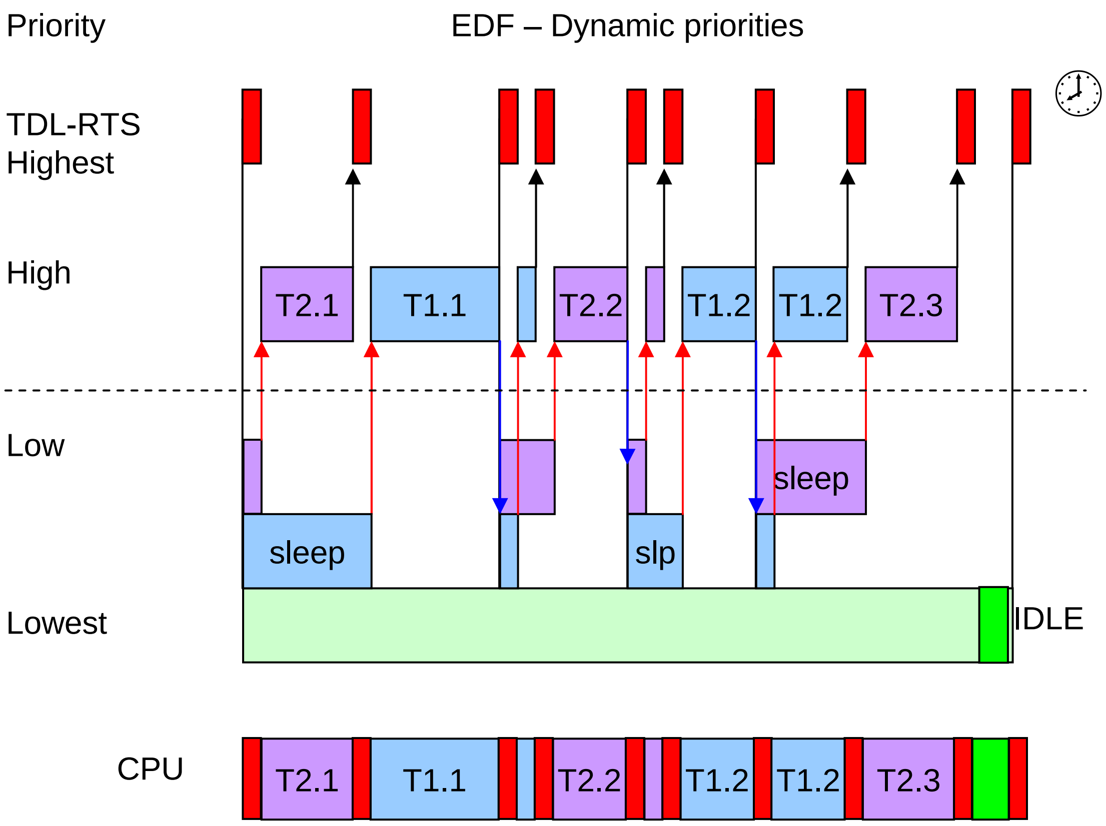
Logical execution of two tasks

Freq.
T1 = 2
T2 = 3



InTime/POSIX – Sample EDF scheduling

- Thread priority changes allow implementation of EDF

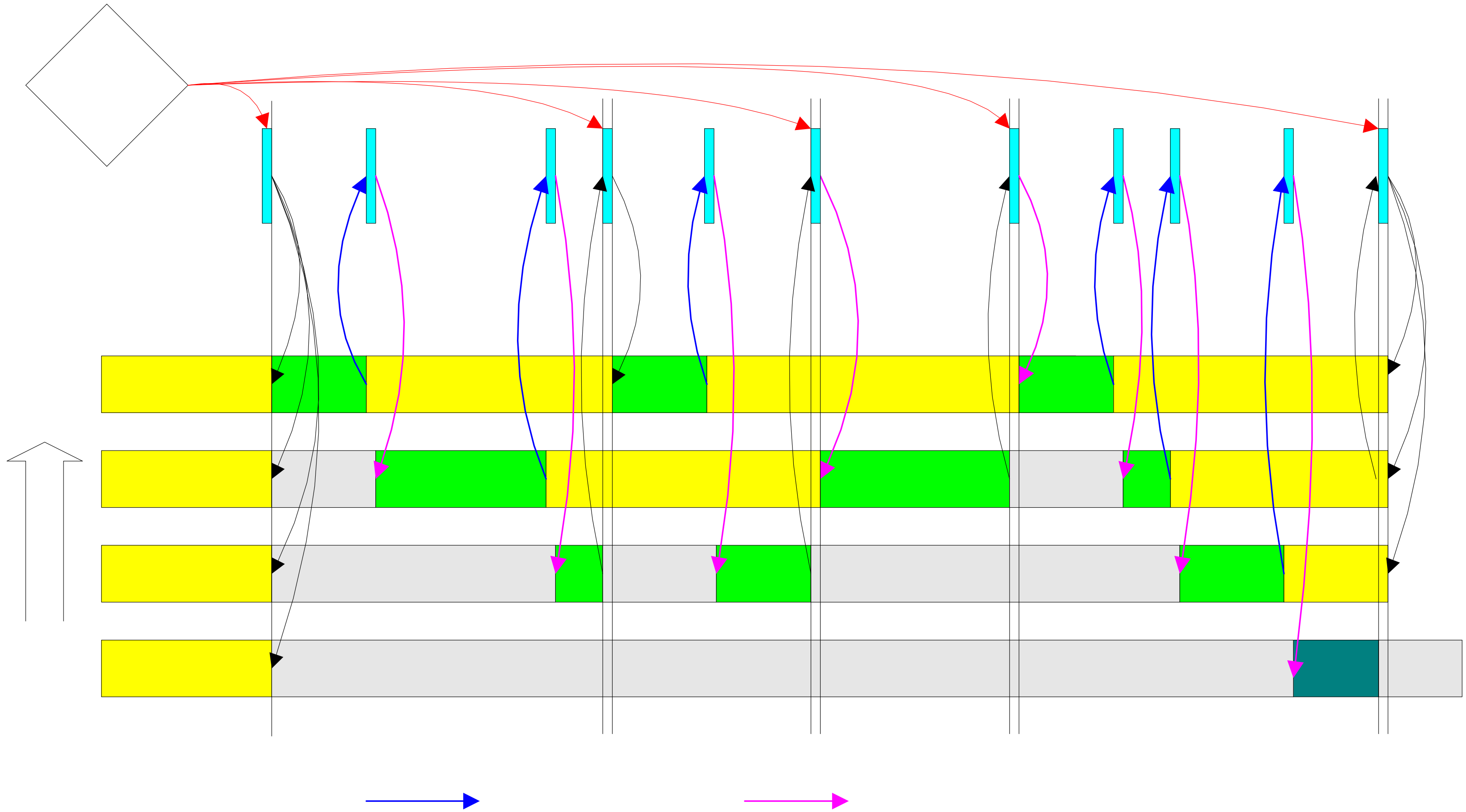


UNIVERSITÄT SALZBURG

RM Scheduling with OSEK

- TDL-RTS
  - non preemptable highest priority task using Alarm signal
  - only **releases** tasks at the right moment **without any real scheduling**
- Tasks
  - basic tasks that have the **priority based on their frequency** (high frequency = high priority) and are fully preemptable
  - end with ChainTask(Emachine) so Emachine can run IDLE or other task
- OSEK Scheduler is used to dispatch tasks based on their priority resulting an RM scheduler

OSEK – Sample RM scheduling



OSEK
Alarm

UNIVERSITÄT SALZBURG

TDL Compiler Plugin – OSEK Specials

- Generates Ecode from TDL data structures
  - Static allocation of Ecode
- Generates corresponding header file
- Generates OIL file containing Tasks, Alarms, Counters, ISR definitions
- Setup Task Launchers
  - Use PIO port to signal task release events
  - Chaining returns to Emachine for scheduling

OSEK Configuration files

```
/***************************/
/*  Functional tasks       */
/***************************/
TASK TDL_Launcher_M1_dec {
   TYPE = BASIC;
   SCHEDULE = FULL;
   PRIORITY = 0;
   ACTIVATION = 1;
   AUTOSTART = FALSE;
   STACKSIZE = 2048;
   SCHEDULE_CALL = FALSE;
};
TASK TDL_Launcher_M1_inc {
   TYPE = BASIC;
   SCHEDULE = FULL;
   PRIORITY = 0;
   ACTIVATION = 1;
   AUTOSTART = FALSE;
   STACKSIZE = 2048;
   SCHEDULE_CALL = FALSE;
};
```

```
CPU KANIS0 {
#include "tdl_comm.oil"
/*  Included Modules                    */
#include "M1.oil"
/*  TDL-RTS core                        */
     TASK _TDL_Emachine_Init {
              TYPE = BASIC;
              SCHEDULE = NON;
              PRIORITY = 0;
              ACTIVATION = 1;
              AUTOSTART = TRUE;
              STACKSIZE = 2048;
              SCHEDULE_CALL = FALSE;
     };
     TASK _TDL_Emachine_Scheduler {
              TYPE = BASIC;
              SCHEDULE = NON;
              PRIORITY = 3;
              ACTIVATION = 5;
              AUTOSTART = FALSE;
              STACKSIZE = 2048;
              SCHEDULE_CALL = FALSE;
     };
     ALARM TimerAlarm {
              COUNTER = SystemTimer;
              ACTION = ACTIVATETASK {
                   TASK = _TDL_Emachine_Scheduler;
              };
     };
     COUNTER SystemTimer {
              MAXALLOWEDVALUE = 65535;
              TICKSPERBASE = 1000;
              MINCYCLE = 1;
     };
     [...]
```

UNIVERSITÄT SALZBURG

Platform specific distribution support

TDL-Comm – OSEK Time Sync

- Use of external time sampler – high resolution, hard to implement, system dependent
  - CAN controller provided timer
  - TPU processor
- Use of internal counter in logical time – lower resolution, OSEK compliant
  - Alarm triggering of tasks and communication
  - Timestamps directly in logical time
  - May drift from real time
  - Increased drift if localtime is reset each cycle

TDL-Comm – OSEK + CAN specific

- Initialize network controller and provide interrupt handler or communication trigger
- Access to remote resources – public ports
  - Use buffers to maintain static properties of the code
- Provide GlobalTime service
  - Select Master Clock – fault tolerance
  - Sync with Master Clock to prevent local time drift
    - Sync each mode period and reset localtime
    - Sync on each slot and maintain absolute timing