ARA: Automatic Instance-Level Analysis in Real-Time Systems

Gerion Entrup, Benedikt Steinmeier, Christian Dietrich

Leibniz Universität Hannover

July 9, 2019

supported by DFG

A Hard Beginning

- Getting a FreeRTOS project from Github:

```
% git clone https://github.com/grafalex82/GPSLogger
Cloning into 'GPSLogger'...
remote: Enumerating objects: 1245, done.
remote: Counting objects: 100% (1245/1245), done.
remote: Compressing objects: 100% (666/666), done.
remote: Total 9544 (delta 683), reused 992 (delta 567), pack-reused 8299
Receiving objects: 100% (9544/9544), 52.33 MiB | 9.47 MiB/s, done.
Resolving deltas: 100% (6615/6615), done.
```

- Repository size: 65 MiB
- 134 000 lines of code

A Hard Beginning

- Getting a FreeRTOS project from Github:

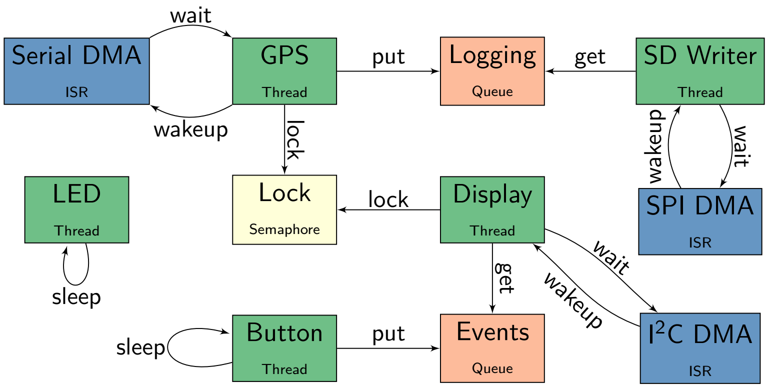
```
% git clone https://github.com/grafalex82/GPSLogger
Cloning into 'GPSLogger'...
remote: Enumerating objects: 1245, done.
remote: Counting objects: 100% (1245/1245), done.
remote: Compressing objects: 100% (666/666), done.
remote: Total 9544 (delta 683), reused 992 (delta 567), pack-reused 8299
Receiving objects: 100% (9544/9544), 52.33 MiB | 9.47 MiB/s, done.
Resolving deltas: 100% (6615/6615), done.
```

- Repository size: 65 MiB
- 134 000 lines of code

What is the systems architecture?

The Instance Graph

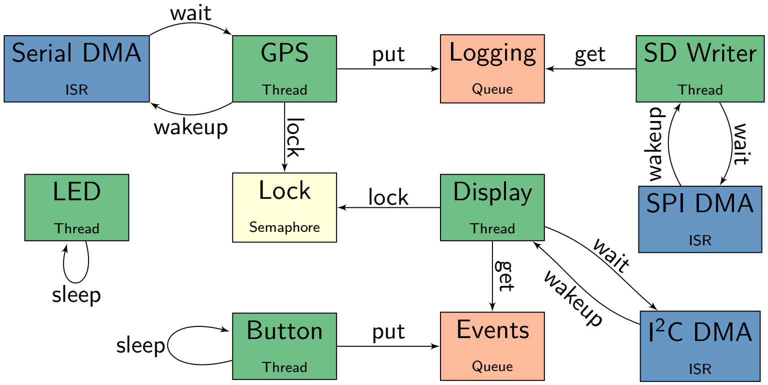
OSPERT'18:
Levels of Specialization in Real-Time Operating Systems



- Get instances of OS abstractions.
- Get interactions between them.

The Instance Graph

OSPERT'18:
Levels of Specialization in Real-Time Operating Systems



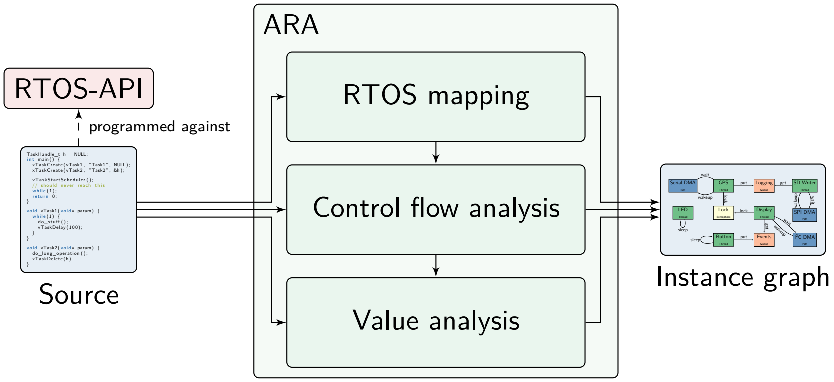
We have extracted the graph manually!
Not possible for larger code bases. We need automation!

Automatic **R**eal-Time Systems **A**nalyzer (ARA)

- Automatic instance graph extraction
- Static source code analysis
  - Application as input
- Supports multiple RTOS interfaces.
  (currently FreeRTOS and OSEK/AUTOSAR)
- Fields of use:
  - System overview
  - Knowledge extraction for specialization
  - OS-API usage validation

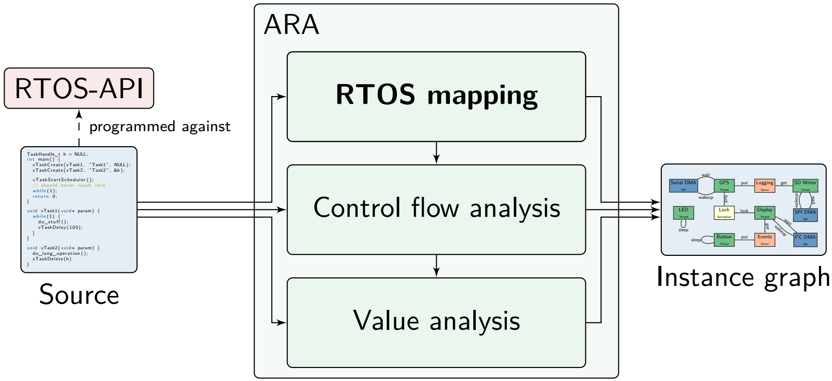
Agenda

ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

OSEK/AUTOSAR vs. FreeRTOS

OSEK/AUTOSAR

.oil

```
TASK t1 {
  PRIORITY = 1;
  SCHEDULE = FULL;
  AUTOSTART = TRUE;
}

TASK t2 {
  PRIORITY = 2;
  SCHEDULE = FULL;
}
```

.cpp

```
TASK(t1) {
  ActivateTask(t2);
}

TASK(t2) {
  TerminateTask();
}
```

FreeRTOS

.cpp

```
TaskHandle_t t1, t2;

int main() {
  t1 = xTaskCreate(task_1, 2);
  t2 = xTaskCreate(task_2, 1);
  vTaskStartScheduler();
}

task_1 { // priority: 2
  vTaskNotifyGive(t1);
}

task_2 { // priority: 1
  while (true) {
    ulTaskNotifyTake();
    vTaskDelete(NULL);
  }
}
```

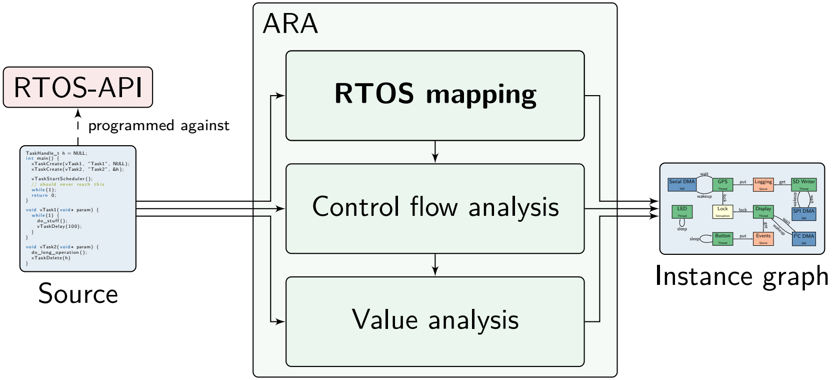
RTOS Mapping

- Detect all **system calls**
- Create unified model

```
# OSEK
"ActivateTask":               (os_type.activate, ...)
"TerminateTask":              (os_type.destroy, ...)
"GetResource":                (os_type.take, ...)
"ReleaseResource":            (os_type.commit, ...)
# FreeRTOS
"xTaskCreate":                (os_type.create, ...)
"vTaskNotifyGive":            (os_type.commit, ...)
"ulTaskNotifyTake" :          (os_type.take, ...)
"xQueueTakeMutexRecursive":   (os_type.take, ...)
"xQueueGiveMutexRecursive":   (os_type.commit, ...)
```

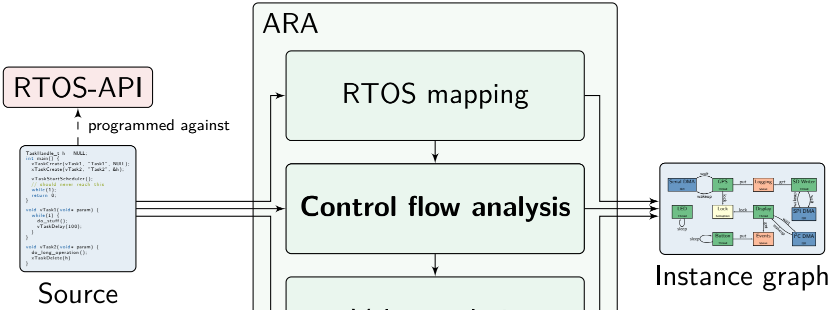
- Create parser for extra data (like OIL file).

ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

ARA in a Nutshell

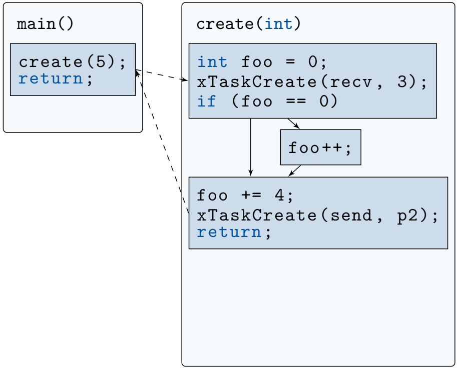


```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

System-Call Aware ICFG

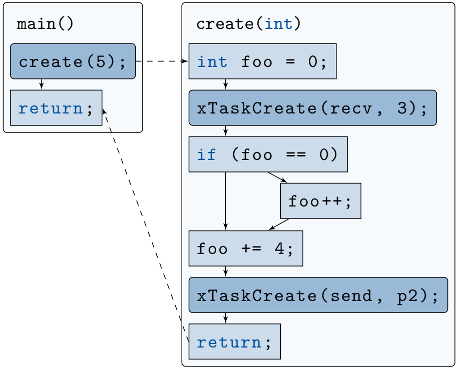
1. Extract interprocedural control flow graph (with LLVM).

```
void recv();
void send();

void create(int p2) {
  int foo = 0;
  xTaskCreate(recv, 3);
  if (foo == 0)
    foo++;
  foo += 4;
  xTaskCreate(send, p2);
  return;
}

int main() {
  create(5);
  return;
}
```



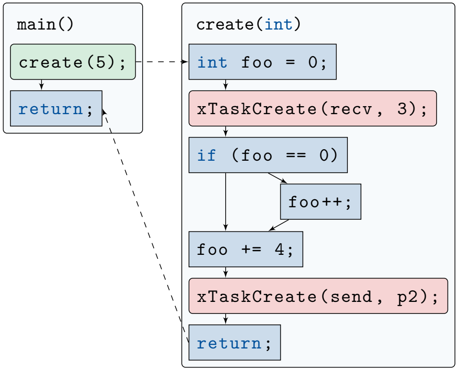
System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).
2. Split calls in separate blocks.



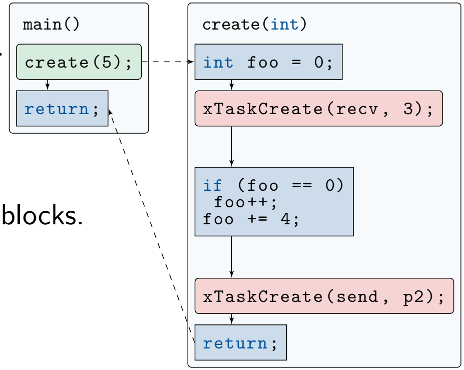
System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).
2. Split calls in separate blocks.
3. Label block types.
   system call, call, computation

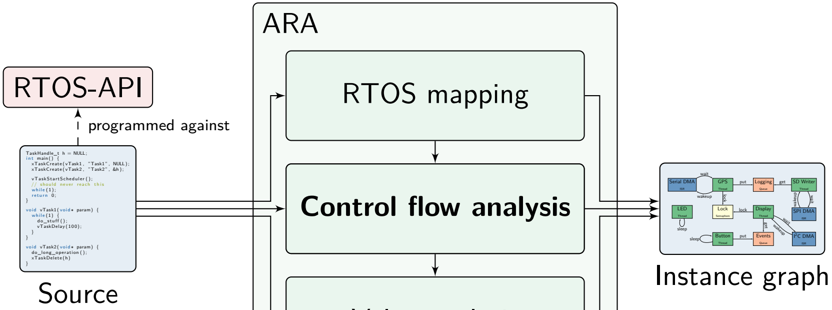


System-Call Aware ICFG

1. Extract interprocedural control flow graph (with LLVM).
2. Split calls in separate blocks.
3. Label block types.
   system call, call, computation
4. Merge appropriate computation blocks.

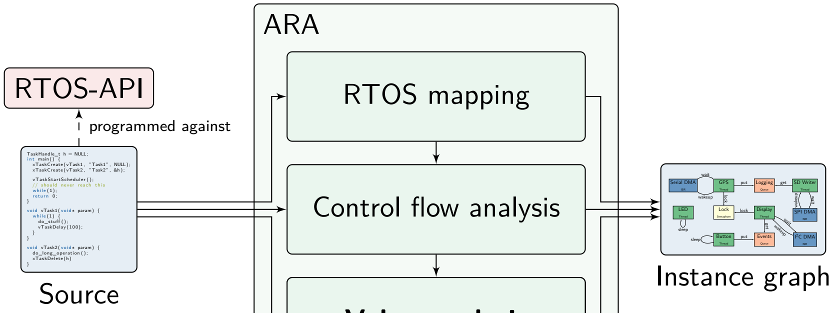


ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

Value Analysis

- Get arguments for system calls.
- Backward search from the call site.
- Follow def-use chain.
- Follow callee-caller relationship.
- Take unambiguous values.

```
void recv();
void send();

void create(int p2) {
  int foo = 0;
  xTaskCreate(recv, 3);
  if (foo == 0)
    foo++;
  foo += 4;
  xTaskCreate(send, p2);
  return;
}

int main() {
  create(5);
  return;
}
```

Value Analysis

- Get arguments for system calls.
- Backward search from the call site.
- Follow def-use chain.
- Follow callee-caller relationship.
- Take unambiguous values.

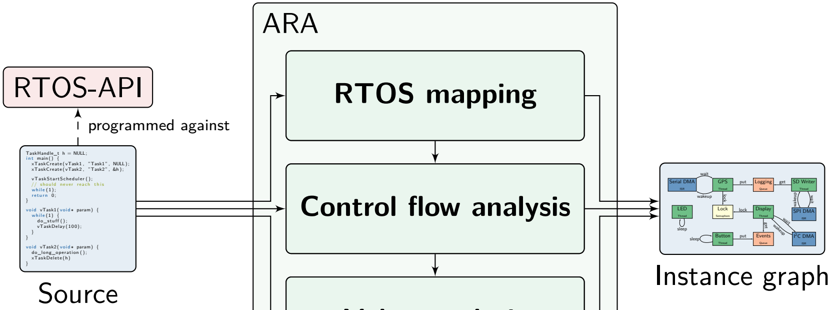
```
void recv();
void send();

void create(int p2) {
  int foo = 0;
  xTaskCreate(recv, 3);
  if (foo == 0)
    foo++;
  foo += 4;
  xTaskCreate(send, p2);
  return;
}

int main() {
  create(5);
  return;
}
```

Value Analysis

- Get arguments for system calls.
- Backward search from the call site.
- Follow def-use chain.
- Follow callee-caller relationship.
- Take unambiguous values.

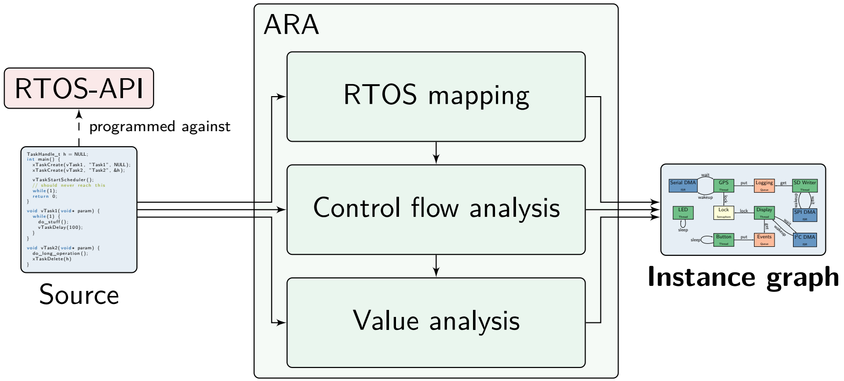
```
void recv();
void send();

void create(int p2) {
  int foo = 0;
  xTaskCreate(recv, 3);
  if (foo == 0)
    foo++;
  foo += 4;
  xTaskCreate(send, p2);
  return;
}

int main() {
  create(5);
  return;
}
```

ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

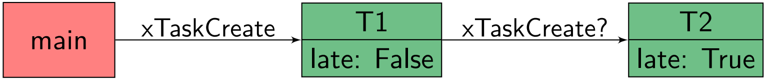
ARA in a Nutshell



```
TaskHandle_t h = NULL;
int main() {
  xTaskCreate(vTask1, "Task1", NULL);
  xTaskCreate(vTask2, "Task2", &h);

  vTaskStartScheduler();
  // should never reach this
  while(1);
  return 0;
}

void vTask1(void* param) {
  while(1) {
    do_stuff();
    vTaskDelay(100);
  }
}

void vTask2(void* param) {
  do_long_operation();
  xTaskDelete(h)
}
```

Instance Graph Creation

- Instance creation in branch or loop?
  - ARA marks them with "?".
- Instance creation before or after scheduler start?
  - Before: Only runs once.
  - After: Unknown number of runs.
  - ARA sets "*late*" attribute.



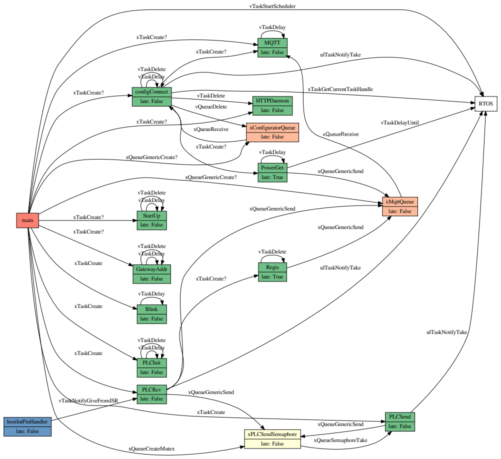
Agenda

Experiments

- Show viability of approach.
- Tested with 4 real-world systems:
  - GPSLogger (FreeRTOS)
  - SmartPlug[1] (FreeRTOS)
  - I4Copter with events (OSEK)
  - I4Copter without events (OSEK)
- Implemented three validation tests:
  - FreeRTOS: Only ISR-capable system calls used in ISRs?
  - OSEK: Does OIL-file match the source code?
  - FreeRTOS/OSEK: Enter and exit of critical region always pairwise?

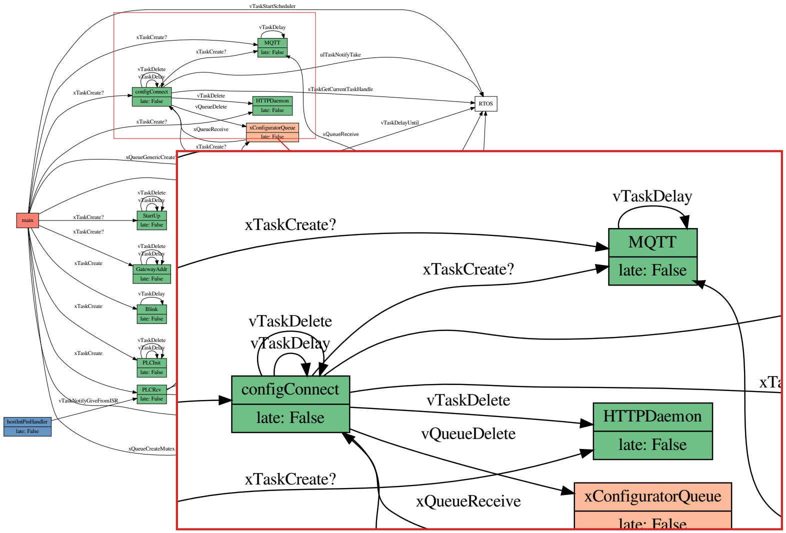
[1] `https://github.com/KKoovalsky/Smartplug`

SmartPlug



vTaskStartScheduler
vTaskDelay
MQTT
late: False
xTaskCreate?
xTaskCreate?
ulTaskNotifyTake
vTaskDelete
vTaskDelay
configConnect
late: False
xTaskCreate?
xTaskGetCurrentTaskHandle
vTaskDelete
HTTPDaemon
late: False
vQueueDelete
RTOS
xTaskCreate?
xConfiguratorQueue
late: False
xQueueReceive
vTaskDelayUntil
xQueueReceive
xTaskCreate?
xQueueGenericCreate?
vTaskDelay
PowerGet
late: True
xQueueGenericCreate?
xQueueGenericSend
vTaskDelete
vTaskDelay
xMqttQueue
late: False
main
xTaskCreate?
StartUp
late: False
xQueueGenericSend
xTaskCreate?
xQueueGenericSend
vTaskDelete
vTaskDelay
vTaskDelete
Regis
late: True
xTaskCreate
GatewayAddr
late: False
ulTaskNotifyTake
xTaskCreate?
vTaskDelay
xTaskCreate
Blink
late: False
ulTaskNotifyTake
vTaskDelete
vTaskDelay
xTaskCreate
PLCInit
late: False
PLCRcv
late: False
xQueueGenericSend
vTaskNotifyGiveFromISR
xTaskCreate
hostIntPinHandler
late: False
PLCSend
late: False
xQueueGenericSend
xPLCSendSemaphore
late: False
xQueueSemaphoreTake
xQueueCreateMutex

SmartPlug

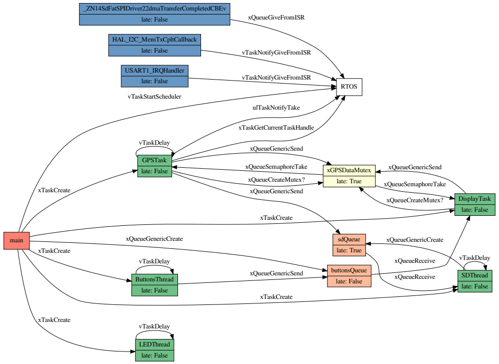


vTaskStartScheduler
xTaskCreate?
vTaskDelay
MQTT
late: False
xTaskCreate?
ulTaskNotifyTake
vTaskDelete
vTaskDelay
xTaskCreate?
configConnect
late: False
xTaskGetCurrentTaskHandle
vTaskDelete
HTTPDaemon
late: False
vQueueDelete
RTOS
xTaskCreate?
xQueueReceive
xConfiguratorQueue
late: False
vTaskDelayUntil
xQueueReceive
xTaskCreate?
xQueueGenericCreate
main
vTaskDelete
vTaskDelay
xTaskCreate?
StartUp
late: False
xTaskCreate?
vTaskDelete
vTaskDelay
xTaskCreate
GatewayAddr
late: False
vTaskDelay
xTaskCreate
Blink
late: False
vTaskDelete
vTaskDelay
xTaskCreate
PLCInit
late: False
PLCRcv
late: False
vTaskNotifyGiveFromISR
hostIntPinHandler
late: False
xQueueCreateMutex
vTaskDelay
xTaskCreate?
MQTT
late: False
xTaskCreate?
vTaskDelete
vTaskDelay
configConnect
late: False
vTaskDelete
HTTPDaemon
late: False
vQueueDelete
xTaskCreate?
xQueueReceive
xConfiguratorQueue
late: False

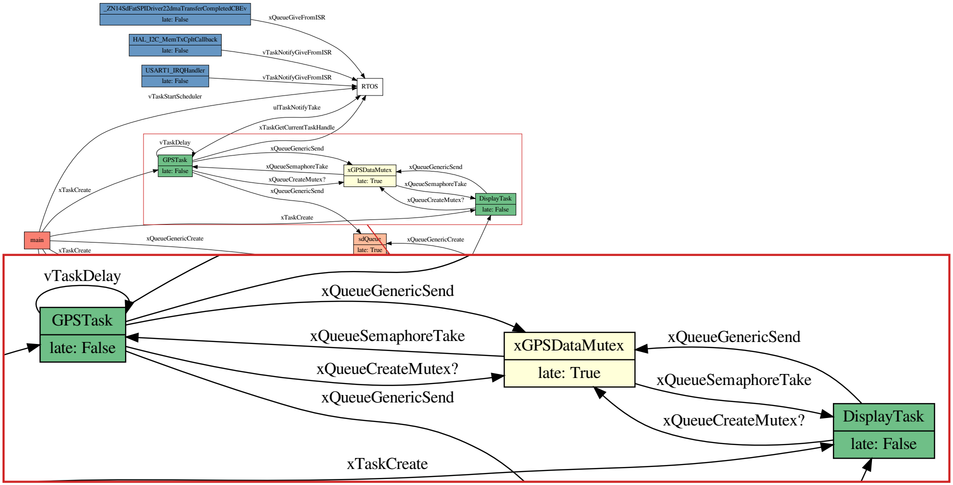
ISE SRA

GPSLogger

Leibniz Universität Hannover



GPSLogger



_ZN14SdFatSPIDriver22dmaTransferCompletedCBEv
late: False
HAL_I2C_MemTxCpltCallback
late: False
USART1_IRQHandler
late: False
xQueueGiveFromISR
vTaskNotifyGiveFromISR
vTaskNotifyGiveFromISR
RTOS
vTaskStartScheduler
ulTaskNotifyTake
xTaskGetCurrentTaskHandle
vTaskDelay
GPSTask
late: False
xQueueGenericSend
xQueueSemaphoreTake
xQueueCreateMutex?
xQueueGenericSend
xGPSDataMutex
late: True
xQueueGenericSend
xQueueSemaphoreTake
xQueueCreateMutex?
DisplayTask
late: False
xTaskCreate
xTaskCreate
main
xQueueGenericCreate
xTaskCreate
sdQueue
late: True
xQueueGenericCreate

Future Work

- Build a global control flow graph (GCFG) [DHL17].
  - Include scheduler decisions.
- Improve value analysis.
  - Alias analysis.
  - Model ambiguous values.
- Interactive graph browsing.
  - Link source code and instance graph.

Conclusion

- ARA[2]
  - Automatic extraction of an instance graph.
  - Supports multiple RTOS interfaces.
  - Show viability with 4 real-world applications.
- Fields of use:
  - Application architecture overview.
  - Knowledge extraction for specialization.
  - OS-API usage validation.

[2] `https://github.com/luhsra/ara`

Conclusion

- ARA[2]
  - Automatic extraction of an instance graph.
  - Supports multiple RTOS interfaces.
  - Show viability with 4 real-world applications.
- Fields of use:
  - Application architecture overview.
  - Knowledge extraction for specialization.
  - OS-API usage validation.

Thank you! Questions?

[2] `https://github.com/luhsra/ara`

References I

Christian Dietrich, Martin Hoffmann, and Daniel Lohmann. "Global Optimization of Fixed-Priority Real-Time Systems by RTOS-Aware Control-Flow Analysis". In: *ACM Transactions on Embedded Computing Systems* 16.2 (2017), 35:1–35:25. DOI: 10.1145/2950053.