BTF Trace Framework on RTFParallella

Generated by Doxygen 1.8.13

Thu Aug 20 2020 12:19:11

Contents

1	Clas	s Index			1
	1.1	Class	List		 1
2	File	Index			3
	2.1	File Lis	st		 3
3	Clas	s Docu	mentation		5
	3.1	Amalth	neaTask_t	Struct Reference	 5
		3.1.1	Detailed	Description	 5
		3.1.2	Member	Data Documentation	 5
			3.1.2.1	cInHandler	 5
			3.1.2.2	cOutHandler	 6
			3.1.2.3	deadline	 6
			3.1.2.4	executionTime	 6
			3.1.2.5	period	 6
			3.1.2.6	src_id	 6
			3.1.2.7	src_instance	 6
			3.1.2.8	task_id	 6
			3.1.2.9	task_instance	 6
			3.1.2.10	taskHandler	 7
	3.2	btf_tra	ce_data_t	Struct Reference	 7
		3.2.1	Member	Data Documentation	 7
			3.2.1.1	data	 7
			3212	eventState	7

ii CONTENTS

		3.2.1.3	eventTypeId	. 7
		3.2.1.4	srcld	. 8
		3.2.1.5	srcInstance	. 8
		3.2.1.6	taskld	. 8
		3.2.1.7	taskInstance	. 8
		3.2.1.8	ticks	. 8
3.3	btf_tra	ce_entity_	_entry_t Struct Reference	. 8
	3.3.1	Member	Data Documentation	. 9
		3.3.1.1	entity_id	. 9
		3.3.1.2	entity_name	. 9
		3.3.1.3	entity_type	. 9
		3.3.1.4	instance	. 9
		3.3.1.5	state	. 9
3.4	btf_tra	ce_entity_	_table_t Struct Reference	. 10
	3.4.1	Member	Data Documentation	. 10
		3.4.1.1	entity_data	. 10
		3.4.1.2	is_occupied	. 10
3.5	btf_tra	ce_heade	er_config_t Struct Reference	. 10
	3.5.1	Detailed	Description	. 11
	3.5.2	Member	Data Documentation	. 11
		3.5.2.1	creator	. 11
		3.5.2.2	modelfile	. 11
		3.5.2.3	timescale	. 11
		3.5.2.4	timeunit	. 11
3.6	btf_tra	ce_info_t	Struct Reference	. 12
	3.6.1	Detailed	Description	. 12
	3.6.2	Member	Data Documentation	. 12
		3.6.2.1	core_id	. 12
		3.6.2.2	core_write	. 12
		3.6.2.3	length	. 12

CONTENTS

		3.6.2.4	offset		 	 	 	 	 	 	. 12
3.7	DSHM	l_section_t	t Struct Refere	ence	 	 	 	 	 	 	. 13
	3.7.1	Detailed	Description		 	 	 	 	 	 	. 13
	3.7.2	Member	Data Docume	entation .	 	 	 	 	 	 	. 13
		3.7.2.1	base_addr		 	 	 	 	 	 	. 13
		3.7.2.2	col		 	 	 	 	 	 	. 13
		3.7.2.3	label_count		 	 	 	 	 	 	. 13
		3.7.2.4	row		 	 	 	 	 	 	. 13
		3.7.2.5	sec_type .		 	 	 	 	 	 	. 14
3.8	labelVi	isual_t Stru	uct Reference		 	 	 	 	 	 	. 14
	3.8.1	Member	Data Docume	entation .	 	 	 	 	 	 	. 14
		3.8.1.1	col		 	 	 	 	 	 	. 14
		3.8.1.2	num_visible	lobola							. 14
				_iabels .	 	 	 	 	 	 	
		3.8.1.3	row					 	 	 	. 14
3.9	SHM_				 	 					
3.9	SHM_s	section_t S	row		 	 	 	 	 	 	. 15
3.9		section_t S Detailed	row Struct Referen		 	 	 	 	 	 	. 15
3.9	3.9.1	section_t S Detailed	row Struct Referen Description	nce	 	 	 	 	 	 	. 15
3.9	3.9.1	section_t S Detailed Member	row Struct Referen Description Data Docume	entation.	 		 	 	 	 	. 15 . 15 . 15

iv CONTENTS

4	File	Docum	entation		17
	4.1	Amalth	ieaConver	rter.h File Reference	17
		4.1.1	Detailed	Description	18
		4.1.2	Macro D	efinition Documentation	18
			4.1.2.1	numTasks	18
			4.1.2.2	PLATFORM_WORD_LENGTH	18
		4.1.3	Typedef	Documentation	18
			4.1.3.1	AmaltheaTask	19
		4.1.4	Function	Documentation	19
			4.1.4.1	calculateStackSize()	19
			4.1.4.2	createAmaItheaTask()	19
			4.1.4.3	createRTOSTask()	20
			4.1.4.4	generalizedRTOSTask()	20
	4.2	c2c.h F	File Refere	ence	21
		4.2.1	Detailed	Description	22
		4.2.2	Typedef	Documentation	22
			4.2.2.1	DSHM_section	23
		4.2.3	Function	Documentation	23
			4.2.3.1	allocate_epiphany_memory()	23
			4.2.3.2	DSHM_section_init()	23
			4.2.3.3	get_base_address_core()	24
			4.2.3.4	read_DSHM_section()	24
			4.2.3.5	shared_label_read_core()	24
			4.2.3.6	shared_label_write_core()	25
			4.2.3.7	shared_labels_init_core()	25
			4.2.3.8	write_DSHM_section()	26
	4.3	debugl	Flags.h Fil	e Reference	26
		4.3.1	Detailed	Description	28
		4.3.2	Function	Documentation	28
			4.3.2.1	get_time_scale_factor()	28

CONTENTS

		4.3.2.2	init_btf_mem_section()	28
		4.3.2.3	init_task_trace_buffer()	29
		4.3.2.4	signalHost()	29
		4.3.2.5	traceRunningTask()	29
		4.3.2.6	traceTaskEvent()	30
		4.3.2.7	traceTaskPasses()	30
		4.3.2.8	updateDebugFlag()	30
		4.3.2.9	updateTick()	31
4.4	FreeR	TOSConfig	g.h File Reference	31
	4.4.1	Detailed	Description	32
4.5	host_u	ıtils.h File F	Reference	33
	4.5.1	Detailed	Description	34
4.6	label_r	man_core0	O.h File Reference	34
	4.6.1	Detailed	Description	34
4.7	model	_enumerat	tions.h File Reference	35
	4.7.1	Detailed	Description	36
	4.7.2	Function	Documentation	36
		4.7.2.1	generate_hw_entity_table()	36
		4.7.2.2	generate_runnable_entity_table()	36
		4.7.2.3	generate_signal_entity_table()	37
		4.7.2.4	generate_task_entity_table()	37
		4.7.2.5	get_DSHM_label_name()	37
		4.7.2.6	get_SHM_label_name()	38
		4.7.2.7	get_task_name()	38
		4.7.2.8	get_visible_label_index()	38
4.8	Paralle	ellaUtils.h F	File Reference	39
	4.8.1	Detailed	Description	40
	4.8.2	Function	Documentation	40
		4.8.2.1	sleepTimerMs()	40
4.9	RTFPa	arallellaCor	nfig.h File Reference	40

vi CONTENTS

	4.9.1	Detailed Description	42
	4.9.2	Typedef Documentation	42
		4.9.2.1 btf_trace_info	42
	4.9.3	Enumeration Type Documentation	42
		4.9.3.1 TYPE	42
	4.9.4	Variable Documentation	43
		4.9.4.1 execution_time_scale	43
4.10	shared	_comms.h File Reference	43
	4.10.1	Detailed Description	45
	4.10.2	Typedef Documentation	45
		4.10.2.1 SHM_section	45
	4.10.3	Function Documentation	45
		4.10.3.1 allocate_shared_memory()	45
		4.10.3.2 read_shm_section()	46
		4.10.3.3 shm_section_init()	46
		4.10.3.4 shm_section_init_read()	46
		4.10.3.5 write_shm_section()	47
4.11	taskCo	de.h File Reference	47
	4.11.1	Detailed Description	48
4.12	trace_u	tils_BTF.h File Reference	48
	4.12.1	Detailed Description	50
	4.12.2	Typedef Documentation	51
		4.12.2.1 btf_trace_header_config_t	51
	4.12.3	Enumeration Type Documentation	51
		4.12.3.1 btf_trace_event_name_t	51
		4.12.3.2 btf_trace_event_type_t	51
	4.12.4	Function Documentation	52
		4.12.4.1 get_btf_trace_file_path()	52
		4.12.4.2 parse_btf_trace_arguments()	52
		4.12.4.3 store_entity_entry()	53
		4.12.4.4 write_btf_trace_data()	53
		4.12.4.5 write_btf_trace_header_config()	54
		4.12.4.6 write_btf_trace_header_entity_table()	54
		4.12.4.7 write_btf_trace_header_entity_type()	55
		4.12.4.8 write_btf_trace_header_entity_type_table()	55
Index			57

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AmaltheaTask_t													 			 	5
btf_trace_data_t													 				7
btf_trace_entity_entry_t .													 			 	8
btf_trace_entity_table_t .													 			 	10
btf_trace_header_config_t													 			 	10
btf_trace_info_t													 			 	12
DSHM_section_t													 			 	13
labelVisual_t													 			 	14
SHM section t													 			 	15

2 Class Index

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

AmaltheaConverter.h	
This file declares and implements function to generate Amalthea task model. The functions defined in this file are used to generate the Amalthea Task model and create a generalized RTOS task which executes periodically in infinite loop over FreeRTOS	17
c2c.h	
This file declares and implements function to communicate between epiphany cores. The functions defined in this file are used for reading from and writing data in the epiphany cores	21
debugFlags.h	
This file declares the debug trace of the application running on Epiphany core. The debug trace consists in two forms. One is the visual form which is the legacy RTFParallella trace and can be seen while executing the application on Adapteva Parallella. The other is the BTF trace dump which can be viewed on Eclipse Trace Compass	26
FreeRTOSConfig.h	
This file declares the macros and structures used on FreeRTOS Configuration	31
host_utils.h	
This file is used to define the utility functions for the host core application	33
label_man_core0.h	
This file is used to define the functions for implementing the cln and cOut handler and reading and writing of shared and distributed labels and signals	34
model_enumerations.h	
This file declares and implements the entity table for BTF trace generation. It consists of functions used to store the entities in the tracing framework which is used to generate the BTF trace	35
Parallella Utils.h	
This file declares the sleep timer function to simulate the time taken by each task to complete its processing on the hardware core	39
RTFParallellaConfig.h	
This file declares the macros and structures used on Epiphany core to get the trace information,	40
shared_comms.h	
This file declares and implements function to read and write data to shared memory. The functions defined in this file are used for reading from and writing data in the shared memory which	40
can be used by host core or epiphany cores	43
This file is used to define the functions for implementing tasks handlers	47
trace_utils_BTF.h	7/
This file declares and implement the BTF trace framework. It consists of functions used to generate the trace information of the tasks, runnables shared label access and hardware info in	40

File Index

Chapter 3

Class Documentation

3.1 AmaltheaTask_t Struct Reference

#include <AmaltheaConverter.h>

Public Attributes

- unsigned int src_id
- unsigned int src_instance
- · unsigned int task_id
- unsigned int task_instance
- void(* taskHandler)(int src_id, int src_instance)
- unsigned int executionTime
- unsigned int deadline
- unsigned int period
- void(* cInHandler)()
- void(* cOutHandler)()

3.1.1 Detailed Description

Structure to hold tasks according to amalthea model

3.1.2 Member Data Documentation

3.1.2.1 clnHandler

```
void(* AmaltheaTask_t::cInHandler) ()
```

cln handler of the task

6 Class Documentation

```
3.1.2.2 cOutHandler
void(* AmaltheaTask_t::cOutHandler) ()
cOut handler of the task
3.1.2.3 deadline
unsigned int AmaltheaTask_t::deadline
Deadline of the task
3.1.2.4 executionTime
unsigned int AmaltheaTask_t::executionTime
Worst case execution time
3.1.2.5 period
unsigned int AmaltheaTask_t::period
Period of the task
3.1.2.6 src_id
unsigned int AmaltheaTask_t::src_id
Source ID
3.1.2.7 src_instance
unsigned int AmaltheaTask_t::src_instance
Source Instance
3.1.2.8 task_id
unsigned int AmaltheaTask_t::task_id
Task ID
3.1.2.9 task_instance
unsigned int AmaltheaTask_t::task_instance
```

Task Instance

3.1.2.10 taskHandler

```
void(* AmaltheaTask_t::taskHandler) (int src_id, int src_instance)
```

Task handler

The documentation for this struct was generated from the following file:

· AmaltheaConverter.h

3.2 btf_trace_data_t Struct Reference

Public Attributes

- int32 t ticks
- int32_t srcld
- int32_t srcInstance
- int32_t eventTypeId
- int32_t taskld
- int32_t taskInstance
- int32_t eventState
- int32_t data

3.2.1 Member Data Documentation

3.2.1.1 data

```
int32_t btf_trace_data_t::data
```

Notes

3.2.1.2 eventState

```
int32_t btf_trace_data_t::eventState
```

State of the event

3.2.1.3 eventTypeId

```
int32_t btf_trace_data_t::eventTypeId
```

Type of event Runnable, Task etc..

8 Class Documentation

3.2.1.4 srcld

```
int32_t btf_trace_data_t::srcId
```

Source Id

3.2.1.5 srcInstance

```
int32_t btf_trace_data_t::srcInstance
```

Instance of the source

3.2.1.6 taskld

```
int32_t btf_trace_data_t::taskId
```

Task Id

3.2.1.7 taskinstance

```
int32_t btf_trace_data_t::taskInstance
```

Instance of the task

3.2.1.8 ticks

```
int32_t btf_trace_data_t::ticks
```

Tick count

The documentation for this struct was generated from the following file:

• trace_utils_BTF.h

3.3 btf_trace_entity_entry_t Struct Reference

Public Attributes

- uint16_t entity_id
- int16_t instance
- btf_trace_event_name state
- btf_trace_event_type entity_type
- uint8_t entity_name [64]

3.3.1 Member Data Documentation

3.3.1.1 entity_id uint16_t btf_trace_entity_entry_t::entity_id Entity ID to get the entity name 3.3.1.2 entity_name uint8_t btf_trace_entity_entry_t::entity_name[64]

3.3.1.3 entity_type

Entity name

```
\verb|btf_trace_event_type| | \verb|btf_trace_entity_entry_t::entity_type| |
```

Entity type to get the source

3.3.1.4 instance

```
int16_t btf_trace_entity_entry_t::instance
```

Current instance of the entity

3.3.1.5 state

```
btf_trace_event_name btf_trace_entity_entry_t::state
```

Current state of the entity

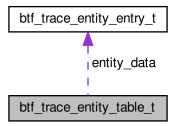
The documentation for this struct was generated from the following file:

• trace_utils_BTF.h

10 Class Documentation

3.4 btf_trace_entity_table_t Struct Reference

Collaboration diagram for btf_trace_entity_table_t:



Public Attributes

- uint16_t is_occupied
- btf_trace_entity_entry entity_data

3.4.1 Member Data Documentation

3.4.1.1 entity_data

btf_trace_entity_entry btf_trace_entity_table_t::entity_data

Entity details

3.4.1.2 is_occupied

uint16_t btf_trace_entity_table_t::is_occupied

If 0, entry is available else not available

The documentation for this struct was generated from the following file:

• trace_utils_BTF.h

3.5 btf_trace_header_config_t Struct Reference

#include <trace_utils_BTF.h>

Public Attributes

- uint32_t timescale
- uint8_t creator [64]
- uint8_t modelfile [512]
- uint8_t timeunit [4]

3.5.1 Detailed Description

Structure to hold BTF Header

3.5.2 Member Data Documentation

3.5.2.1 creator

```
uint8_t btf_trace_header_config_t::creator[64]
```

Target device on which the trace is generated

3.5.2.2 modelfile

```
uint8_t btf_trace_header_config_t::modelfile[512]
```

Model file used to generate the trace

3.5.2.3 timescale

```
uint32_t btf_trace_header_config_t::timescale
```

This sets the scale of the time e.g 10, 100 etc..

3.5.2.4 timeunit

```
uint8_t btf_trace_header_config_t::timeunit[4]
```

Time unit e.g ns, ms, us, s..

The documentation for this struct was generated from the following file:

• trace_utils_BTF.h

12 Class Documentation

3.6 btf_trace_info_t Struct Reference

```
#include <RTFParallellaConfig.h>
```

Public Attributes

- · int length
- · unsigned int offset
- · unsigned int core_id
- unsigned int core_write

3.6.1 Detailed Description

Structure to ensure proper synchronization between host and epiphany cores and also within epiphany cores.

3.6.2 Member Data Documentation

```
3.6.2.1 core_id
unsigned int btf_trace_info_t::core_id
Epiphany core id
```

```
3.6.2.2 core_write
unsigned int btf_trace_info_t::core_write
```

Read write operation between epiphany core and host

```
3.6.2.3 length
int btf_trace_info_t::length
```

To define the length of BTF packets to be read

```
3.6.2.4 offset
unsigned int btf_trace_info_t::offset
```

Defines the offset location in memory area

The documentation for this struct was generated from the following file:

RTFParallellaConfig.h

3.7 DSHM_section_t Struct Reference

#include <c2c.h>

Public Attributes

- unsigned row
- unsigned col
- unsigned int base_addr
- unsigned label_count
- TYPE sec_type

3.7.1 Detailed Description

defines a distributed shared memory section

3.7.2 Member Data Documentation

3.7.2.1 base_addr

unsigned int DSHM_section_t::base_addr

address of the first label in the section

3.7.2.2 col

unsigned DSHM_section_t::col

the column of target core on Epi chip

3.7.2.3 label_count

unsigned DSHM_section_t::label_count

number of labels in the section

3.7.2.4 row

unsigned DSHM_section_t::row

the row of target core on Epi chip

14 Class Documentation

3.7.2.5 sec_type

```
TYPE DSHM_section_t::sec_type
```

data type of the section (size of labels in the section)

The documentation for this struct was generated from the following file:

• c2c.h

3.8 labelVisual_t Struct Reference

Public Attributes

- · unsigned row
- unsigned col
- unsigned num_visible_labels

3.8.1 Member Data Documentation

```
3.8.1.1 col
```

unsigned labelVisual_t::col

Column ID of the hardware core

3.8.1.2 num_visible_labels

unsigned labelVisual_t::num_visible_labels

Number of visual labels

3.8.1.3 row

unsigned labelVisual_t::row

Row ID of the hardware core

The documentation for this struct was generated from the following file:

host_utils.h

3.9 SHM_section_t Struct Reference

```
#include <shared_comms.h>
```

Public Attributes

- unsigned int base_addr
- unsigned label_count
- TYPE sec_type

3.9.1 Detailed Description

defines a shared memory section

3.9.2 Member Data Documentation

```
3.9.2.1 base_addr
```

```
unsigned int SHM_section_t::base_addr
```

address of the first label in the section

3.9.2.2 label_count

```
unsigned SHM_section_t::label_count
```

number of labels in the section

3.9.2.3 sec_type

```
TYPE SHM_section_t::sec_type
```

data type of the section (size of labels in the section)

The documentation for this struct was generated from the following file:

· shared_comms.h

16 **Class Documentation**

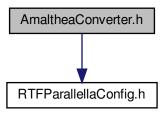
Chapter 4

File Documentation

4.1 AmaltheaConverter.h File Reference

This file declares and implements function to generate Amalthea task model. The functions defined in this file are used to generate the Amalthea Task model and create a generalized RTOS task which executes periodically in infinite loop over FreeRTOS.

#include "RTFParallellaConfig.h"
Include dependency graph for AmaltheaConverter.h:



Classes

struct AmaltheaTask_t

Macros

- #define PLATFORM_WORD_LENGTH 32
- #define numTasks 3

Typedefs

• typedef struct AmaltheaTask_t AmaltheaTask

18 File Documentation

Functions

 AmaltheaTask createAmaltheaTask (void *taskHandler, void *cInHandler, void *cOutHandler, unsigned int period, unsigned int deadline, unsigned int WCET, unsigned int src_id, unsigned int src_instance, unsigned int task_id, unsigned int task_instance)

Generating Amalthea task model.

void createRTOSTask (AmaltheaTask *task, int priority, int argCount,...)

Create the RTOS task that represents a given Amalthea task.

unsigned int calculateStackSize (int labelBitCount, int labelCount)

This function returns the additional stack size (in words) needed for the task to handle its labels.

void generalizedRTOSTask (AmaltheaTask task)

This RTOS task invokes the task handlers and realizes periodic task execution according to Amalthea model.

4.1.1 Detailed Description

This file declares and implements function to generate Amalthea task model. The functions defined in this file are used to generate the Amalthea Task model and create a generalized RTOS task which executes periodically in infinite loop over FreeRTOS.

Author

Mahmoud Bazzal, Anand Prakash

Date

17 April 2020

4.1.2 Macro Definition Documentation

4.1.2.1 numTasks

#define numTasks 3

Defines number of tasks. Not is current use

4.1.2.2 PLATFORM_WORD_LENGTH

```
#define PLATFORM_WORD_LENGTH 32
```

Not is used in current implementation. Defines the platform word size

4.1.3 Typedef Documentation

4.1.3.1 AmaltheaTask

```
typedef struct AmaltheaTask_t AmaltheaTask
```

Structure to hold tasks according to amalthea model

4.1.4 Function Documentation

4.1.4.1 calculateStackSize()

This function returns the additional stack size (in words) needed for the task to handle its labels.

Parameters

in	labelBitCount	: label size in bits
in	labelCount	: number of labels

Returns

: stack size

This function returns the additional stack size (in words) needed for the task to handle its labels

4.1.4.2 createAmaltheaTask()

```
AmaltheaTask createAmaltheaTask (
    void * taskHandler,
    void * cInHandler,
    void * cOutHandler,
    unsigned int period,
    unsigned int deadline,
    unsigned int WCET,
    unsigned int src_id,
    unsigned int task_id,
    unsigned int task_instance)
```

Generating Amalthea task model.

The function takes the input arguments and generates the Amalthea task model which is used to create the RTOS tasks.

Arguments:

20 File Documentation

Parameters

in	taskHandler	: Amalthea task handler
in	cInHandler	: Amalthea cln handler
in	cOutHandler	: Amalthea cOut handler
in	period	: Time period of the task
in	deadline	: Deadline of the task
in	WCET	: Worst case execution time of the task
in	src_id	: Source ID of the tasks
in	src_instance	: Source Instance of the task
in	task_id	: Task ID
in	task_instance	: Task Instance

Returns

: Amalthea task model

Generating Amalthea task model

4.1.4.3 createRTOSTask()

Create the RTOS task that represents a given Amalthea task.

This function can have multiple arguments for all label types used by the task and the number of labels of each type.

Parameters

in	task	: pointer to the AmaltheaTask struct
in	priority	: priority of the task (according to RMS, lowesrt perio has highest priority)
in	argCount	: number of different types of labels used by this task
in	label_type_size	: size (in bits) of label type.
in	label_type_count	number of labels associated with that type.

Returns

: void

Create the RTOS task that represents a given Amalthea task.

4.1.4.4 generalizedRTOSTask()

```
void generalized
RTOSTask ( {\tt AmaltheaTask}~task~)
```

This RTOS task invokes the task handlers and realizes periodic task execution according to Amalthea model.

4.2 c2c.h File Reference 21

Parameters

in	task	: instance of AmaltheaTask structure to be invoked	1
----	------	--	---

Returns

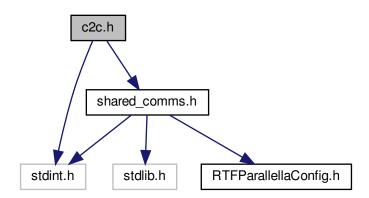
: void

This RTOS task invokes the task handlers and realizes periodic task execution according to Amalthea model

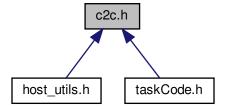
4.2 c2c.h File Reference

This file declares and implements function to communicate between epiphany cores. The functions defined in this file are used for reading from and writing data in the epiphany cores.

```
#include <stdint.h>
#include "shared_comms.h"
Include dependency graph for c2c.h:
```



This graph shows which files directly or indirectly include this file:



22 File Documentation

Classes

• struct DSHM_section_t

Macros

• #define DSHM_SEC_LABEL_COUNT 10

Typedefs

• typedef struct DSHM_section_t DSHM_section

Functions

• unsigned int * allocate_epiphany_memory (unsigned int offset)

Provide the epiphany memory section based on the offset data.

· void shared labels init core (void)

Initiate the shared label section, this function will assign addresses to labels in a section, and initialize those labels to 0

void shared_label_write_core (unsigned row, unsigned col, int label_indx, int payload)

Write a value to a label in a distributed shared memory section.

• unsigned int shared_label_read_core (unsigned row, unsigned col, int label_indx)

Read a value of a label in a distributed shared memory section.

unsigned int get_base_address_core (int row, int col)

Get the absolute base memory address of a core.

• void DSHM_section_init (DSHM_section sec)

Initialize the distributed shared label section.

• unsigned int read DSHM section (DSHM section sec, int label indx)

Read data from a specific label in a distributed shared memory section.

void write_DSHM_section (DSHM_section sec, int label_indx, int payload)

Write data to a specific label in a distributed shared memory section.

4.2.1 Detailed Description

This file declares and implements function to communicate between epiphany cores. The functions defined in this file are used for reading from and writing data in the epiphany cores.

Author

Mahmoud Bazzal, Anand Prakash

Date

17 April 2020

4.2.2 Typedef Documentation

4.2 c2c.h File Reference 23

4.2.2.1 DSHM_section

```
typedef struct DSHM_section_t DSHM_section
```

defines a distributed shared memory section

4.2.3 Function Documentation

4.2.3.1 allocate_epiphany_memory()

Provide the epiphany memory section based on the offset data.

The epiphany memory section is defined in data bank 3 of each core and start at address 0x7000. The address is returned based on the offset to 0x7000 of each core. This buffer is assigned to stored the RTF parallella legacy trace info. Data bank 3 is used to store the information on each epiphany core. It starts at 0x7000 offset on each epiphany core. Any change in this buffer addressing must be followed with the correct offset set in host application to get the correct values.

Parameters

in	offset	: Offset to the epiphany start memory of 0x7000
----	--------	---

Returns

: Pointer to the Epiphany address.

Provide the epiphany memory section based on the offset data.

4.2.3.2 DSHM_section_init()

```
void DSHM_section_init ( {\tt DSHM\_section} \ \ sec )
```

Initialize the distributed shared label section.

Initiate the distributed shared label section, this function will assign addresses to labels in a section, and initialize those labels to the value of 256

Parameters

in	sec	: structure of type DSHM_section containing details of the the distributed shared memory section
		to be initiated

Initialize the distributed shared label section.

24 File Documentation

4.2.3.3 get_base_address_core()

Get the absolute base memory address of a core.

Get he global memory address of the provided epiphany row id and column id.

Parameters

in	row	: absolute row number of the core
in	col	: absolute column number of the core

Returns

: Global memory address of the Epiphany core.

Get the absolute base memory address of a core.

4.2.3.4 read_DSHM_section()

Read data from a specific label in a distributed shared memory section.

This function will read one full label but the result will be cast into unsigned int (4 bytes on this platform) Segmentation fault will occur for addresses outside the shared_dram section of the system check RTFP documentation for details.

Parameters

in	sec	: struct of the section to be read
in	label_indx	: index of requested label

Returns

: value of requested label in a distributed shared memory section

Read data from a specific label in a distributed shared memory section

4.2.3.5 shared_label_read_core()

```
unsigned int shared_label_read_core (
          unsigned row,
          unsigned col,
          int label_indx )
```

Read a value of a label in a distributed shared memory section.

4.2 c2c.h File Reference 25

Parameters

L	in	row	: absolute row number of the core
ſ	in	col	: absolute column number of the core
ſ	in	label_indx	: index of the target shared label

Returns

: Shared label value at the provided label_indx

Read a value of a label in a distributed shared memory section.

4.2.3.6 shared_label_write_core()

```
void shared_label_write_core (
          unsigned row,
          unsigned col,
          int label_indx,
          int payload)
```

Write a value to a label in a distributed shared memory section.

Parameters

in	row	: absolute row number of the core
in	col	: absolute column number of the core
in	label_indx	: index of the target shared label
in	payload	: value to write

Returns

: void

Write a value to a label in a distributed shared memory section

4.2.3.7 shared_labels_init_core()

Initiate the shared label section, this function will assign addresses to labels in a section, and initialize those labels to 0.

Returns

: void

Initiate the shared label section, this function will assign addresses to labels in a section, and initialize those labels to 0

26 File Documentation

4.2.3.8 write_DSHM_section()

Write data to a specific label in a distributed shared memory section.

This function will write one full label but the value will be given as int (4 bytes on this platform) to avoid overflow issues Segmentation fault will occur for addresses outside the shared_dram section of the system check RTFP documentation for details.

Parameters

in	sec	: struct of the section to be written to
in	label_indx	: index of requested label
in	payload	: value to be written (will be cast into data type of target label)

Returns

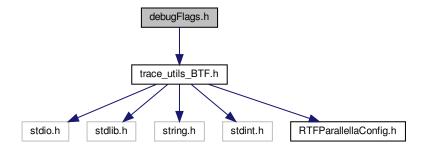
: void

Write data to a specific label in a distributed shared memory section

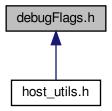
4.3 debugFlags.h File Reference

This file declares the debug trace of the application running on Epiphany core. The debug trace consists in two forms. One is the visual form which is the legacy RTFParallella trace and can be seen while executing the application on Adapteva Parallella. The other is the BTF trace dump which can be viewed on Eclipse Trace Compass.

```
#include "trace_utils_BTF.h"
Include dependency graph for debugFlags.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define cnt_address 0x3000
- #define TASK1_FLAG 2
- #define TASK2_FLAG 0
- #define TASK3 FLAG 4
- #define TASK4_FLAG 0
- #define TASK5 FLAG 4
- #define RUNNINGTASK_FLAG 6
- #define **DEBUG_FLAG** 7
- #define TICK_FLAG 8
- #define BTF_TRACE_BUFFER_SIZE 8

Functions

• unsigned int get_time_scale_factor (void)

Get the time scaling factor for tick count.

void init_btf_mem_section (void)

Initialize memory section for storing BTR trace data and metadata.

void init_task_trace_buffer (void)

Initialize output buffer in core memory.

void traceRunningTask (unsigned taskNum)

Write the index of the running task to output buffer.

void traceTaskPasses (unsigned taskNum, int currentPasses)

write the task instance (job) to output buffer

void updateTick (void)

Update RTOS tick value in output buffer.

void updateDebugFlag (int debugMessage)

Write a custom value to the output buffer for code coverage debugging.

void signalHost (void)

Signal the host core to read the memory.

void traceTaskEvent (int srcID, int srcInstance, btf_trace_event_type type, int taskId, int taskInstance, btf_
 trace_event_name event_name, int data)

Write the BTF trace data.

28 File Documentation

4.3.1 Detailed Description

This file declares the debug trace of the application running on Epiphany core. The debug trace consists in two forms. One is the visual form which is the legacy RTFParallella trace and can be seen while executing the application on Adapteva Parallella. The other is the BTF trace dump which can be viewed on Eclipse Trace Compass.

Author

Mahmoud Bazzal, Anand Prakash

Date

10 April 2020

4.3.2 Function Documentation

```
4.3.2.1 get_time_scale_factor()
```

Get the time scaling factor for tick count.

Returns

: Scale factor

```
4.3.2.2 init_btf_mem_section()
```

Initialize memory section for storing BTR trace data and metadata.

The function does not take any arguments. It initializes the BTF memory section from the shared memory area.

Returns

: void

4.3.2.3 init_task_trace_buffer()

Initialize output buffer in core memory.

The function initializes the epiphany core memory section for dumping the legacy RTParallella trace.

Returns

: void

4.3.2.4 signalHost()

```
void signalHost (
     void )
```

Signal the host core to read the memory.

The function locks the shared memory address using the epiphany mutex implementation. It waits until wait the host core processor has read the data. It then dumps the BTF trace metadata to the shared memory and and unlock the mutex. DMA channel 1 is used to dump the trace metadata.

Returns

: void

Signal the host to read the BTF trace metadata

4.3.2.5 traceRunningTask()

Write the index of the running task to output buffer.

The function writes the ID of the current task in execution to the Epiphany core memory

Parameters

```
in taskNum: index of the task
```

Returns

: void

4.3.2.6 traceTaskEvent()

```
void traceTaskEvent (
    int srcID,
    int srcInstance,
    btf_trace_event_type type,
    int taskId,
    int taskInstance,
    btf_trace_event_name event_name,
    int data )
```

Write the BTF trace data.

The function dumps the BTF trace data to the shared memory.. DMA channel 1 is used to dump the trace data.

Parameters

in	srcID	: Source ID of the task.
in	srcInstance	: Source instance of the task.
in	type	: Event type.
in	taskld	: Task Id.
in	taskInstance	: Task instance.
in	event_name	: Name of the event.
in	data	: Notes or shared label value

Returns

: void

4.3.2.7 traceTaskPasses()

```
void traceTaskPasses (
          unsigned taskNum,
          int currentPasses )
```

write the task instance (job) to output buffer

The function writes the task instance number of the current task in execution to the Epiphany core memory.

Parameters

in	taskNum	: index of the task
in	currentPasses	: instance of task (job number)

4.3.2.8 updateDebugFlag()

```
{\tt void updateDebugFlag (}
```

```
int debugMessage )
```

Write a custom value to the output buffer for code coverage debugging.

Parameters

	in	debugMessage	: message to be written	
--	----	--------------	-------------------------	--

Returns

: void

4.3.2.9 updateTick()

```
void updateTick (
     void )
```

Update RTOS tick value in output buffer.

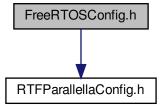
Returns

: void

4.4 FreeRTOSConfig.h File Reference

This file declares the macros and structures used on FreeRTOS Configuration.

```
#include "RTFParallellaConfig.h"
Include dependency graph for FreeRTOSConfig.h:
```



Macros

- #define configCALL_STACK_SIZE 0x50
- #define configUSE PREEMPTION 1
- #define configUSE_TIME_SLICING 0
- #define configUSE IDLE HOOK 0
- #define configUSE TICK HOOK 0
- #define configCPU_CLOCK_HZ ((unsigned long) 700000000)
- #define configTICK_RATE_HZ ((TickType_t) execution_time_scale)
- #define configMAX_PRIORITIES (5)
- #define configMINIMAL STACK SIZE ((unsigned short) 112)
- #define configTOTAL_HEAP_SIZE ((size_t) (10240))
- #define configMAX_TASK_NAME_LEN (48)
- #define configUSE_TRACE_FACILITY 0
- #define configUSE_16_BIT_TICKS 1
- #define configIDLE_SHOULD_YIELD 0
- #define configUSE_ALTERNATIVE_API 0
- #define configUSE_CO_ROUTINES 0
- #define configMAX_CO_ROUTINE_PRIORITIES (2)
- #define INCLUDE_vTaskPrioritySet 0
- #define INCLUDE_uxTaskPriorityGet 0
- #define INCLUDE_vTaskDelete 0
- #define INCLUDE_vTaskCleanUpResources 0
- #define INCLUDE_vTaskSuspend 1
- #define INCLUDE vTaskDelayUntil 1
- #define INCLUDE_vTaskDelay 1
- #define INCLUDE xTaskGetCurrentTaskHandle 1
- #define INCLUDE_pcTaskGetTaskName 1
- · #define C2C_MSG_TYPE int

4.4.1 Detailed Description

This file declares the macros and structures used on FreeRTOS Configuration.

Author

Mahmoud Bazzal, Anand Prakash

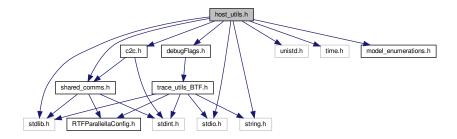
Date

15 April, 2020

4.5 host utils.h File Reference

This file is used to define the utility functions for the host core application.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include "c2c.h"
#include "debugFlags.h"
#include "shared_comms.h"
#include "model_enumerations.h"
Include dependency graph for host utils.h:
```



Classes

struct labelVisual t

Macros

- #define READ PRECISION US 1000
- #define MEM_TYPE_SHM 0
- #define MEM_TYPE_DSHM 1

Typedefs

typedef struct labelVisual_t LabelVisual

Functions

- void array_init (unsigned array[], unsigned array_size)
- void user_config_print_legend (LabelVisual core_config, unsigned array[])
- void user_config_print_values (LabelVisual core_config, unsigned array[], unsigned int values_array[], unsigned int prv_val_array[])
- LabelVisual get_user_input (unsigned indices[])
- void user_config_print_legend_auto (unsigned array_length, unsigned array[])
- void **user_config_print_values_auto** (unsigned visible_label_count, unsigned array[], unsigned int values_array[], unsigned int prv_val_array[])
- unsigned get user input DRAM (unsigned indices[])
- void **print_legend_enum** (unsigned label_count, unsigned label_positions[], unsigned memory_type)
- int nsleep (long miliseconds)

4.5.1 Detailed Description

This file is used to define the utility functions for the host core application.

Author

Mahmoud Bazzal, Anand Prakash

Date

24 May 2020

4.6 label_man_core0.h File Reference

This file is used to define the functions for implementing the cln and cOut handler and reading and writing of shared and distributed labels and signals.

Macros

• #define num_unique_sections 1

Functions

- void init_mem_sections (void)
- void init_DSHM_sections (void)
- void cln5ms ()
- void cln10ms ()
- void cln20ms ()
- void cln10msCore2 ()
- void cln20msCore2 ()
- void cOut5ms ()
- · void cOut10ms ()
- · void cOut20ms ()
- void cOut10msCore2 ()
- void cOut20msCore2 ()

4.6.1 Detailed Description

This file is used to define the functions for implementing the cln and cOut handler and reading and writing of shared and distributed labels and signals.

Author

Mahmoud Bazzal, Anand Prakash

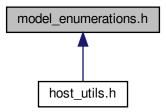
Date

24 May 2020

4.7 model enumerations.h File Reference

This file declares and implements the entity table for BTF trace generation. It consists of functions used to store the entities in the tracing framework which is used to generate the BTF trace.

This graph shows which files directly or indirectly include this file:



Macros

- #define SHM LABEL COUNT 10
- #define DSHM_LABEL_COUNT 10
- #define LABEL_STRLEN 32
- #define **EXEC_TASK_COUNT** 5
- #define EXEC_CORE_COUNT 2
- #define SHM_VISIBLE_LABEL_COUNT 2
- #define **DSHM_VISIBLE_LABEL_COUNT** 2

Functions

• void get_SHM_label_name (int index, char *str)

Get the string name of DRAM shared label.

void get_DSHM_label_name (int index, char *str)

Get the string name of distributed shared label (on a core)

void get_task_name (int index, char *str)

Get the string name of the task being run.

• void get_visible_label_index (unsigned array[], unsigned mem_type)

Get the indices of required labels to show in either shared memory or distributed shared memory.

void generate_task_entity_table (void)

Generate the BTF trace entity entry for all the tasks.

· void generate runnable entity table (void)

Generate the BTF trace entity entry for all the runnables.

void generate_signal_entity_table (void)

Generate the BTF trace entity entry for all the label/signal entities.

void generate_hw_entity_table (void)

Generate the BTF trace entity entry for all the hardware entities.

4.7.1 Detailed Description

This file declares and implements the entity table for BTF trace generation. It consists of functions used to store the entities in the tracing framework which is used to generate the BTF trace.

Author

Mahmoud Bazzal, Anand Prakash

Date

20 May 2020

4.7.2 Function Documentation

4.7.2.1 generate_hw_entity_table()

Generate the BTF trace entity entry for all the hardware entities.

The function is used to store all the hardware entities used in the tasks execution on a heterogeneous platform which is used to generate the BTF header and data section.

Returns

: void

Generate the BTF trace entity entry for all the hardware entities

4.7.2.2 generate_runnable_entity_table()

Generate the BTF trace entity entry for all the runnables.

The function is used to store all the runnable entities used in the tasks execution on a heterogeneous platform which is used to generate the BTF header and data section.

Returns

: void

Generate the BTF trace entity entry for all the runnables

4.7.2.3 generate_signal_entity_table()

Generate the BTF trace entity entry for all the label/signal entities.

The function is used to store all the shared and distributed label entities used in the tasks execution on a heterogeneous platform which is used to generate the BTF header and data section.

Returns

: void

Generate the BTF trace entity entry for all the label/signal entities

4.7.2.4 generate_task_entity_table()

```
void generate_task_entity_table ( \mbox{void} \ \ \mbox{)}
```

Generate the BTF trace entity entry for all the tasks.

The function is used to store all the tasks entities used in the tasks execution on a heterogeneous platform which is used to generate the BTF header and data section.

Returns

: void

Generate the BTF trace entity entry for all the tasks

4.7.2.5 get_DSHM_label_name()

Get the string name of distributed shared label (on a core)

Parameters

in	index	: shared label index in the memory section
in,out	*str	: pointer to buffer string that holds the name

Returns

: void

4.7.2.6 get_SHM_label_name()

```
void get_SHM_label_name (
                int index,
                char * str )
```

Get the string name of DRAM shared label.

Parameters

in	index	: shared label index in the shared memory section
in,out	*str	: pointer to buffer string that holds the name

Returns

: void

4.7.2.7 get_task_name()

```
void get_task_name (
          int index,
          char * str )
```

Get the string name of the task being run.

Parameters

in	index	: task index in the task_enum array
in,out	*str	: pointer to buffer string that holds the name

Returns

: void

Get the string name of the task being run

4.7.2.8 get_visible_label_index()

```
void get_visible_label_index (
          unsigned array[],
          unsigned mem_type )
```

Get the indices of required labels to show in either shared memory or distributed shared memory.

Parameters

in,out	array	: array buffer that holds the indices
in	mem_type	: the memory type of indices requested (MEM_TYPE_SHM or MEM_TYPE_DSHM)

Returns

: void

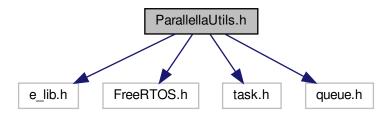
Get the indices of required labels to show in either shared memory or distributed shared memory

4.8 ParallellaUtils.h File Reference

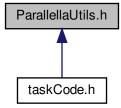
This file declares the sleep timer function to simulate the time taken by each task to complete its processing on the hardware core.

```
#include "e_lib.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
```

Include dependency graph for ParallellaUtils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define _1MS 700000
- #define _1US 700

Functions

void sleepTimerMs (int ticks, int taskNum)

Sleep for a multiple of milliseconds.

4.8.1 Detailed Description

This file declares the sleep timer function to simulate the time taken by each task to complete its processing on the hardware core.

Author

Mahmoud Bazzal, Anand Prakash

Date

20 May 2020

4.8.2 Function Documentation

4.8.2.1 sleepTimerMs()

```
void sleepTimerMs (
    int ticks,
    int taskNum )
```

Sleep for a multiple of milliseconds.

It makes the task to sleep for the provided millisecond. This is used to simulate the computation time taken by the task in a real scenario.

Parameters

in	ticks	: number of milliseconds to sleep
in	taskNum	: index of task invoking this function (used for tracing during sleep)

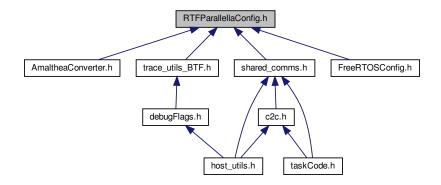
Returns

: void

4.9 RTFParallellaConfig.h File Reference

This file declares the macros and structures used on Epiphany core to get the trace information,.

This graph shows which files directly or indirectly include this file:



Classes

· struct btf trace info t

Macros

- #define SHARED DRAM START ADDRESS 0x8E000000
- #define SHARED_DRAM_START_OFFSET 0x01000000
- #define **SHARED_DRAM_SECTION** (SHARED_DRAM_START_ADDRESS + SHARED_DRAM_START ← OFFSET)
- #define SHARED_DRAM_SIZE 0x00002000
- #define RTF_DEBUG_TRACE_COUNT 10
- #define INPUT_TIMESCALE_OFFSET 20
- #define SHARED_BTF_DATA_OFFSET (INPUT_TIMESCALE_OFFSET + 4)
- #define SHARED_LABEL_OFFSET 0x1000
- #define BTF_TRACE_BUFFER_SIZE 8
- #define GLOBAL_SHARED_LABEL_OFFSET sizeof(btf_trace_info)
- #define SHM_LABEL_COUNT 10
- #define DSHM_LABEL_CORE_OFFSET 10
- #define ECORE_RTF_BUFFER_ADDR 0x7000
- #define DSHM_LABEL_EPI_CORE_OFFSET 0x7040
- #define MUTEX ROW 1
- #define MUTEX_COL 0
- #define RING BUFFER SIZE 6

Typedefs

- typedef struct btf_trace_info_t btf_trace_info
- typedef enum entity_id_t entity_id

Enumerations

```
    enum entity_id_t {
        IDLE_TASK_ID = 0, TASK5MS0_ID, TASK10MS0_ID, TASK20MS0_ID,
        TASK10MS1_ID, TASK20MS1_ID, RUNNABLE_HANDLER5MS0_ID = 16, RUNNABLE_HANDLER10
        MS0_ID,
        RUNNABLE_HANDLER20MS0_ID, RUNNABLE_HANDLER10MS1_ID, RUNNABLE_HANDLER20MS1
        __ID, SH_LABEL_A_ID = 64,
        SH_LABEL_B_ID, DSH_LABEL_A_ID, DSH_LABEL_B_ID, HW_CORE0_ID = 256,
        HW_CORE1_ID }
    enum TYPE { UINT_8, UINT_16, UINT_32 }
```

Variables

· unsigned int execution_time_scale

4.9.1 Detailed Description

This file declares the macros and structures used on Epiphany core to get the trace information,.

Author

Anand Prakash

Date

19 June 2020

4.9.2 Typedef Documentation

```
4.9.2.1 btf trace info
```

```
typedef struct btf_trace_info_t btf_trace_info
```

Structure to ensure proper synchronization between host and epiphany cores and also within epiphany cores.

4.9.3 Enumeration Type Documentation

4.9.3.1 TYPE

enum TYPE

Enumerator

UINT_8	unsigned char type
UINT_16	unsigned short type
UINT_32	unsigned int type

4.9.4 Variable Documentation

4.9.4.1 execution_time_scale

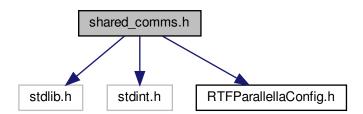
unsigned int execution_time_scale

Time scale factor per tick

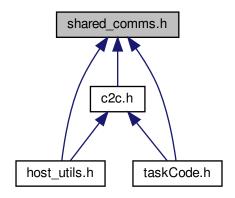
4.10 shared_comms.h File Reference

This file declares and implements function to read and write data to shared memory. The functions defined in this file are used for reading from and writing data in the shared memory which can be used by host core or epiphany cores.

```
#include <stdlib.h>
#include <stdint.h>
#include "RTFParallellaConfig.h"
Include dependency graph for shared_comms.h:
```



This graph shows which files directly or indirectly include this file:



Classes

struct SHM_section_t

Macros

- #define shared_mem_section 0x01001000
- #define shared_mem_section_2 0x01001000
- #define shared_mem_section1_label_count 10
- #define SHM_DEFINED_SPACE 256

Typedefs

• typedef struct SHM section t SHM section

Functions

unsigned int * allocate_shared_memory (unsigned int offset)

The function provides the pointer to the global address to the shared DRAM memory.

void * shm_section_init (SHM_section sec)

Function to initialize the shared memory area.

• int read shm section (unsigned int *x, unsigned indx)

Read data from a specific label in a shared memory section.

void write_shm_section (unsigned int *x, unsigned indx, int payload)

Write data to a specific label in a shared memory section.

• unsigned int shm_section_init_read (SHM_section sec, int index)

This function is obsolete. Use "read_shm_section" for reading shared memory area.

4.10.1 Detailed Description

This file declares and implements function to read and write data to shared memory. The functions defined in this file are used for reading from and writing data in the shared memory which can be used by host core or epiphany cores.

Author

Mahmoud Bazzal, Anand Prakash

Date

13 April 2020

4.10.2 Typedef Documentation

4.10.2.1 SHM_section

```
typedef struct SHM_section_t SHM_section
```

defines a shared memory section

4.10.3 Function Documentation

4.10.3.1 allocate_shared_memory()

The function provides the pointer to the global address to the shared DRAM memory.

The shared DRAM memory offset starts at 0x8F000018. This address space is accessible by the Epiphany cores as well as Host. The actual DRAM starts at 0x8F000000. The first 20 bytes are reserved by the FreeRTOS. The next 4 bytes is used to store the time scale. The next 44 bytes will be used to store the BTF trace information. The rest of the memory can be used for storing the shared labels.

Parameters

in	offset	: The offset from the shared dram start address.

Returns

: Pointer to the shared DRAM memory.

The function provides the pointer to the global address to the shared DRAM memory.

4.10.3.2 read_shm_section()

```
int read_shm_section (
          unsigned int * x,
           unsigned indx )
```

Read data from a specific label in a shared memory section.

This function will read one full label but the result will be cast into int (4 bytes on this platform)

Segmentation fault will occur for addresses outside the shared_dram section of the system check RTFP documentation for details.

Parameters

in	X	: pointer to the section to be read
in	indx	: index of requested label

Returns

: value of requested label in a shared memory section

Read data from a specific label in a shared memory section.

4.10.3.3 shm_section_init()

```
void* shm_section_init ( {\tt SHM\_section} \ \ sec \ )
```

Function to initialize the shared memory area.

Initiate the shared label section, this function will assign addresses to labels in a section, and initialize those labels to the value of 256 If the requested section does not fit in the system's shared_dram, a null pointer will be returned

Parameters

in	sec	: structure of type SHM_section containing details of the the shared memory section to be initiated
	500	. Structure of type of mi_scotion containing details of the the shared memory scotion to be initiated

Returns

: pointer to the initiated shared memory label

Function to initialize the shared memory area.

4.10.3.4 shm_section_init_read()

This function is obsolete. Use "read_shm_section" for reading shared memory area.

TODO use e_write/ e_read functions and DMA to constrain contention from different cores.

4.10.3.5 write_shm_section()

```
void write_shm_section (
          unsigned int * x,
          unsigned indx,
          int payload )
```

Write data to a specific label in a shared memory section.

This function will write one full label but the value will be given as int (4 bytes on this platform) to avoid overflow issues

Segmentation fault will occur for addresses outside the shared_dram section of the system check RTFP documentation for details.

Parameters

	in	: pointer to the section to be written to	
in indx : index of requested label		indx	: index of requested label
	in	payload	: value to be written (will be cast into data type of target label)

Returns

: void

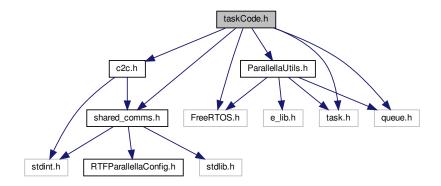
Write data to a specific label in a shared memory section.

4.11 taskCode.h File Reference

This file is used to define the functions for implementing tasks handlers.

```
#include "c2c.h"
#include "ParallellaUtils.h"
#include "shared_comms.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
```

Include dependency graph for taskCode.h:



Functions

- void handler5ms (int src_id, int src_instance)
- void handler10ms (int src_id, int src_instance)
- void handler20ms (int src_id, int src_instance)
- · void handler10msCore2 (int src_id, int src_instance)
- void handler20msCore2 (int src_id, int src_instance)

4.11.1 Detailed Description

This file is used to define the functions for implementing tasks handlers.

Author

Mahmoud Bazzal, Anand Prakash

Date

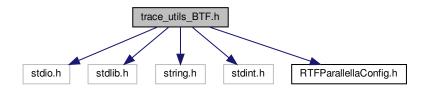
24 May 2020

4.12 trace_utils_BTF.h File Reference

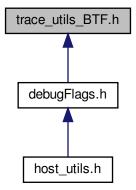
This file declares and implement the BTF trace framework. It consists of functions used to generate the trace information of the tasks, runnables shared label access and hardware info in the BTF trace format.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

#include "RTFParallellaConfig.h"
Include dependency graph for trace_utils_BTF.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct btf_trace_header_config_t
- struct btf_trace_entity_entry_t
- struct btf_trace_data_t
- struct btf_trace_entity_table_t

Macros

- #define RFTP_GENERATE_BTF_TRACE 0x01
- #define **BTF_TRACE_TRUE** 0x01
- #define BTF_TRACE_FALSE 0x00
- #define BTF_TRACE_BUFFER_SIZE 8
- #define TIME_FLAG 0
- #define SOURCE FLAG 1
- #define SOURCE INSTANCE FLAG 2
- #define **EVENT_TYPE_FLAG** 3
- #define TARGET_FLAG 4
- #define TARGET_INSTANCE_FLAG 5
- #define EVENT_FLAG 6
- #define DATA_FLAG 7

Typedefs

- typedef enum btf_trace_event_type_t btf_trace_event_type
- · typedef enum btf trace event name t btf trace event name
- typedef struct btf_trace_header_config_t btf_trace_header_config_t
- · typedef struct btf trace entity entry t btf trace entity entry
- typedef struct btf_trace_data_t btf_trace_data
- · typedef struct btf trace entity table t btf trace entity table

Enumerations

```
    enum btf_trace_event_type_t {
        TASK_EVENT, INT_SERVICE_ROUTINE_EVENT, RUNNABLE_EVENT, INS_BLOCK_EVENT,
        STIMULUS_EVENT, ECU_EVENT, PROCESSOR_EVENT, CORE_EVENT,
        SCHEDULER_EVENT, SIGNAL_EVENT, SEMAPHORE_EVENT, SIMULATION_EVENT }
    enum btf_trace_event_name_t {
        INIT = -1, PROCESS_START, PROCESS_TERMINATE, PROCESS_PREEMPT,
```

PROCESS_SUSPEND, PROCESS_RESUME, SIGNAL_READ, SIGNAL_WRITE }

Functions

• void get_btf_trace_file_path (char *trace_file_path)

Function to get the file name of the trace file along with the absolute path.

int parse_btf_trace_arguments (int argc, char **argv)

Parse the command line arguments for generating the BTF trace file.

void write_btf_trace_header_config (FILE *stream)

This function is responsible for writing the BTF trace header information.

void write_btf_trace_header_entity_type (FILE *stream, btf_trace_event_type type)

This function to write entity type in BTF header data.

void write_btf_trace_header_entity_table (FILE *stream)

Function to write entity type in BTF header data.

void write_btf_trace_header_entity_type_table (FILE *stream)

This function writes the entity type table in the BTF header.

void store_entity_entry (entity_id typeId, btf_trace_event_type type, const char *name)

This function is used to store the entity information of all the tasks, runnables and labels.

void write_btf_trace_data (FILE *stream, uint8_t core_id, unsigned int *data_buffer)

Function to write the data section of the BTF.

4.12.1 Detailed Description

This file declares and implement the BTF trace framework. It consists of functions used to generate the trace information of the tasks, runnables shared label access and hardware info in the BTF trace format.

Author

Anand Prakash

Date

23 May 2020

See also

```
https://wiki.eclipse.org/images/e/e6/TA_BTF_Specification_2.1.3_Eclipse← _Auto_IWG.pdf
```

4.12.2 Typedef Documentation

4.12.2.1 btf_trace_header_config_t

```
typedef struct btf_trace_header_config_t btf_trace_header_config_t
```

Structure to hold BTF Header

4.12.3 Enumeration Type Documentation

4.12.3.1 btf_trace_event_name_t

enum btf_trace_event_name_t

Enumerator

INIT	Dummy Init Event name
PROCESS_START	Starting a process/task/runnable
PROCESS_TERMINATE	Terminating a process/task/runnable
PROCESS_PREEMPT	Preempt a process/task
PROCESS_SUSPEND	Suspend a runnable event
PROCESS_RESUME	Resume a process/task/runnable
SIGNAL_READ	Read event for signal/label
SIGNAL_WRITE	Write signal for signal/label

4.12.3.2 btf_trace_event_type_t

enum btf_trace_event_type_t

Enumerator

TASK_EVENT	BTF Task Event
INT_SERVICE_ROUTINE_EVENT	BTF ISR Event
RUNNABLE_EVENT	BTF Runnable Event
INS_BLOCK_EVENT	BTF INS Block Event
STIMULUS_EVENT	BTF Stimulus Event
ECU_EVENT	BTF ECU Event
PROCESSOR_EVENT	BTF Processor Event
CORE_EVENT	BTF Hardware Core Event
SCHEDULER_EVENT	BTF Scheduler Event

Enumerator

SIGNAL_EVENT	BTF Signal Event for reading/writing labels
SEMAPHORE_EVENT	BTF Semaphore Event
SIMULATION_EVENT	BTF Simulation Event

4.12.4 Function Documentation

4.12.4.1 get_btf_trace_file_path()

Function to get the file name of the trace file along with the absolute path.

Arguments:

Parameters

in, out trace_file_path : Pointer to the be	uffer where the BTF trace file path is stored.
---	--

Returns

: void

Function to get the file name of the trace file along with the absolute path.

4.12.4.2 parse_btf_trace_arguments()

Parse the command line arguments for generating the BTF trace file.

The provided parameters are used to configure the trace file required to be generated. For example the trace file path, model file used to generate the trace, device name and time scale.

Arguments:

Parameters

in	argc	: The count for the number of arguments passed
in	argv	: Pointer to the list of arguments

Returns

: The integer value of the timescale used for the task execution.

Parse the command line arguments for generating the BTF trace file

4.12.4.3 store_entity_entry()

This function is used to store the entity information of all the tasks, runnables and labels.

Store the entity metadata which can be used to generate the entity type and entity type table. Also this table entry is used to decode the tasks and runnables information received from the Parallella framework.

Arguments:

Parameters

in	type← Id	: Unique entity type ID
in	type	: Entity typee.g TASK, RUNNABLE etc
in	name	: Entity name

Returns

: void

This function is used to store the entity information of all the tasks, runnables and labels.

4.12.4.4 write_btf_trace_data()

```
void write_btf_trace_data (
          FILE * stream,
          uint8_t core_id,
          unsigned int * data_buffer )
```

Function to write the data section of the BTF.

The function is responsible for writing the BTF trace data section in CSV format which can be interpreted by the trace visualizing tools such as Eclipse trace compass. Currently the support is provided for only two cores. However, this can be extended further for multiple cores.

Arguments:

Parameters

in	stream	: File pointer to the stream where the data has to be written.
in	core_id : Core ID on which the task operations are performed	
in data_buffer : Da		: Data buffer containing the BTF trace information.

Generated on Thu Aug 20 2020 12:19:11 for BTF Trace Framework on RTFParallella by Doxygen

Returns

: void

Function to write the data section of the BTF

```
4.12.4.5 write_btf_trace_header_config()
```

```
void write_btf_trace_header_config ( {\tt FILE} \, * \, stream \, \, )
```

This function is responsible for writing the BTF trace header information.

Function to write BTF header data to the trace file. It writes the version, creator, input model file, time scale and timestamp section of the header file. It also writes the entity table, type table and entity type table used in the task model.

Arguments:

Parameters

in	stream	: File pointer to the stream where the data has to be written.
----	--------	--

Returns

: void

This function is responsible for writing the BTF trace header information.

```
4.12.4.6 write_btf_trace_header_entity_table()
```

Function to write entity type in BTF header data.

Arguments:

Parameters

in	stream	: File pointer to the stream where the data has to be written.
----	--------	--

Returns

: void

Function to write entity type in BTF header data

4.12.4.7 write_btf_trace_header_entity_type()

This function to write entity type in BTF header data.

The function defines what kinds of entities are supported in the BTF trace generated. It consists of entity type such as Tasks, Signals, Runnables along with their IDs. Refer to below link for more details: $https://wiki.epsiloneeclipse.org/images/e/e6/TA_BTF_Specification_2.1.3_Eclipse_Auto_IWG.pdf$

Arguments:

Parameters

in	stream	: File pointer to the stream where the data has to be written.
in	type	: Type of the entity i.e. TASK, RUNNABLE, STIMULUS etc

Returns

: void

This function to write entity type in BTF header data.

4.12.4.8 write_btf_trace_header_entity_type_table()

This function writes the entity type table in the BTF header.

The function writes the list of tasks, runnables, shared labels, cores in a tabular format. It consists of the tasks, runnables and shared labels executed on the specified cores along with their IDs. Refer to below link for more details: https://wiki.eclipse.org/images/e/e6/TA_BTF_Specification_2.1.3

_Eclipse_Auto_IWG.pdf

Arguments:

Parameters

	in	stream	: File pointer to the stream where the data has to be written.
--	----	--------	--

Returns

: void

This function writes the entity type table in the BTF header.

Index

allocate_epiphany_memory	trace_utils_BTF.h, 51
c2c.h, <mark>23</mark>	btf_trace_header_config_t, 10
allocate_shared_memory	creator, 11
shared_comms.h, 45	modelfile, 11
AmaltheaConverter.h, 17	timescale, 11
AmaltheaTask, 18	timeunit, 11
calculateStackSize, 19	trace_utils_BTF.h, 51
createAmaltheaTask, 19	btf_trace_info
createRTOSTask, 20	RTFParallellaConfig.h, 42
generalizedRTOSTask, 20	btf_trace_info_t, 12
numTasks, 18	core_id, 12
PLATFORM WORD LENGTH, 18	core_write, 12
AmaltheaTask	length, 12
AmaltheaConverter.h, 18	offset, 12
AmaltheaTask_t, 5	
cInHandler, 5	c2c.h, 21
cOutHandler, 5	allocate_epiphany_memory, 23
deadline, 6	DSHM_section, 22
executionTime, 6	DSHM_section_init, 23
period, 6	get_base_address_core, 23
src_id, 6	read_DSHM_section, 24
src_instance, 6	shared_label_read_core, 24
task_id, 6	shared_label_write_core, 25
task_instance, 6	shared_labels_init_core, 25
taskHandler, 6	write_DSHM_section, 25
·	cInHandler
base_addr	AmaltheaTask_t, 5
DSHM_section_t, 13	cOutHandler
SHM_section_t, 15	AmaltheaTask_t, 5
btf_trace_data_t, 7	calculateStackSize
data, 7	AmaltheaConverter.h, 19
eventState, 7	col
eventTypeId, 7	DSHM_section_t, 13
srcld, 7	labelVisual_t, 14
srcInstance, 8	core_id
taskld, 8	btf_trace_info_t, 12
taskInstance, 8	core_write
ticks, 8	btf_trace_info_t, 12
btf_trace_entity_entry_t, 8	createAmaltheaTask
entity_id, 9	AmaltheaConverter.h, 19
entity_name, 9	createRTOSTask
entity_type, 9	AmaltheaConverter.h, 20
instance, 9	creator
state, 9	btf_trace_header_config_t, 11
btf_trace_entity_table_t, 10	
entity_data, 10	DSHM_section
is_occupied, 10	c2c.h, 22
btf_trace_event_name_t	DSHM_section_init
trace_utils_BTF.h, 51	c2c.h, 23
btf trace event type t	DSHM section t. 13

58 INDEX

base_addr, 13	get_time_scale_factor
col, 13	debugFlags.h, 28
label_count, 13	get_visible_label_index
row, 13	model_enumerations.h, 38
sec_type, 13	
data	host_utils.h, 33
btf_trace_data_t, 7	
deadline	init_btf_mem_section
AmaltheaTask t, 6	debugFlags.h, 28
debugFlags.h, 26	init_task_trace_buffer
	debugFlags.h, 28
get_time_scale_factor, 28	instance
init_btf_mem_section, 28	btf_trace_entity_entry_t, 9
init_task_trace_buffer, 28	is_occupied
signalHost, 29	btf_trace_entity_table_t, 10
traceRunningTask, 29	bti_trace_critity_table_t, 10
traceTaskEvent, 29	label_count
traceTaskPasses, 30	DSHM_section_t, 13
updateDebugFlag, 30	SHM_section_t, 15
updateTick, 31	label_man_core0.h, 34
entity_data	labelVisual_t, 14
btf_trace_entity_table_t, 10	col, 14
entity id	num_visible_labels, 14
btf_trace_entity_entry_t, 9	row, 14
entity_name	length
btf_trace_entity_entry_t, 9	btf_trace_info_t, 12
entity_type	
btf_trace_entity_entry_t, 9	model_enumerations.h, 35
eventState	generate_hw_entity_table, 36
	generate_runnable_entity_table, 36
btf_trace_data_t, 7	generate_signal_entity_table, 36
eventTypeId	generate_task_entity_table, 37
btf_trace_data_t, 7	get_DSHM_label_name, 37
execution_time_scale	get_SHM_label_name, 37
RTFParallellaConfig.h, 43	get task name, 38
executionTime	get_visible_label_index, 38
AmaltheaTask_t, 6	modelfile
	btf trace header config t, 11
FreeRTOSConfig.h, 31	bii_iracc_ncacci_comig_i, 11
	num visible labels
generalizedRTOSTask	labelVisual t, 14
AmaltheaConverter.h, 20	numTasks
generate_hw_entity_table	AmaltheaConverter.h, 18
model_enumerations.h, 36	Amaimea Converter.ii, To
generate_runnable_entity_table	offset
model_enumerations.h, 36	btf trace info t, 12
generate_signal_entity_table	bii_iiace_iiiio_i, 12
model_enumerations.h, 36	PLATFORM_WORD_LENGTH
generate_task_entity_table	AmaltheaConverter.h, 18
model_enumerations.h, 37	ParallellaUtils.h, 39
get_DSHM_label_name	
model_enumerations.h, 37	sleepTimerMs, 40
get_SHM_label_name	parse_btf_trace_arguments
model_enumerations.h, 37	trace_utils_BTF.h, 52
	period
get_base_address_core	AmaltheaTask_t, 6
c2c.h, 23	DTEDII-II-C (C. L. 10
get_btf_trace_file_path	RTFParallellaConfig.h, 40
trace_utils_BTF.h, 52	btf_trace_info, 42
get_task_name	execution_time_scale, 43
model_enumerations.h, 38	TYPE, 42

INDEX 59

read_DSHM_section	AmaltheaTask_t, 6
c2c.h, 24	taskld
read_shm_section	btf_trace_data_t, 8
shared_comms.h, 46	taskInstance
row	btf trace data t, 8
DSHM_section_t, 13	ticks
labelVisual t, 14	btf_trace_data_t, 8
	timescale
SHM section	btf trace header config t, 11
shared_comms.h, 45	timeunit
SHM_section_t, 15	btf_trace_header_config_t, 11
base_addr, 15	trace_utils_BTF.h, 48
label_count, 15	
sec_type, 15	btf_trace_event_name_t, 51
— · ·	btf_trace_event_type_t, 51
Sec_type	btf_trace_header_config_t, 51
DSHM_section_t, 13	get_btf_trace_file_path, 52
SHM_section_t, 15	parse_btf_trace_arguments, 52
shared_comms.h, 43	store_entity_entry, 53
allocate_shared_memory, 45	write_btf_trace_data, 53
read_shm_section, 46	write_btf_trace_header_config, 54
SHM_section, 45	write_btf_trace_header_entity_table, 54
shm_section_init, 46	write_btf_trace_header_entity_type, 54
shm_section_init_read, 46	write_btf_trace_header_entity_type_table, 55
write_shm_section, 47	traceRunningTask
shared_label_read_core	debugFlags.h, 29
c2c.h, 24	traceTaskEvent
shared_label_write_core	debugFlags.h, 29
c2c.h, 25	traceTaskPasses
shared_labels_init_core	debugFlags.h, 30
c2c.h, 25	acceg. agen, cc
shm_section_init	updateDebugFlag
shared_comms.h, 46	debugFlags.h, 30
shm_section_init_read	updateTick
shared comms.h, 46	debugFlags.h, 31
signalHost	acceg. agen, c
debugFlags.h, 29	write_DSHM_section
sleepTimerMs	c2c.h, 25
ParallellaUtils.h, 40	write_btf_trace_data
	trace_utils_BTF.h, 53
src_id AmaltheaTask t, 6	write_btf_trace_header_config
- ·	trace_utils_BTF.h, 54
src_instance AmaltheaTask t, 6	write_btf_trace_header_entity_table
	trace utils BTF.h, 54
srcid	write btf trace header entity type
btf_trace_data_t, 7	trace_utils_BTF.h, 54
srcInstance	write_btf_trace_header_entity_type_table
btf_trace_data_t, 8	trace utils BTF.h, 55
state	
btf_trace_entity_entry_t, 9	write_shm_section
store_entity_entry	shared_comms.h, 47
trace_utils_BTF.h, 53	
7.40	
TYPE	
RTFParallellaConfig.h, 42	
task_id	
AmaltheaTask_t, 6	
task_instance	
AmaltheaTask_t, 6	
taskCode.h, 47	
taskHandler	