CoRTOS:
A minimal OS for Ajit

Anshuman Dhuliya, Prof. Madhav Desai • 2021.12.01

Overview

Objective

Design a functionally complete programming environment for application design and its runtime on the Ajit processor.

Major Features

- Supports Multi-Core Ajit Processor
- Supports synchronization and message passing queues between software threads.
- Programmable trap/interrupt handling.

Current Status

Provides a functionally complete set of features.

Dependency

A complete Ajit development environment setup

Programing with CoRTOS

Create a project with a config.yaml

A project in CoRTOS is a folder with all the C/C++/Assembly files and a `config.yaml` file that describes the project.

Use the API

CoRTOS, along with Ajit's Access Routines provides many convenient APIs to help with application development.

Refer the CoRTOS examples

Simple programs that demonstrate most CoRTOS features.

Creating a Project

A Project and its config.yaml

Project Directory Layout

A user project directory in CoRTOS is a folder with all the C/C++/Assembly files and a `config.yaml` file that describes the project.

config.yaml

This file describes the specification of the target hardware and the software configuration of the project.

*Refer to examples in CoRTOS.

config.yaml - using yaml syntax (short yaml tutorial)

Describes Hardware

- Processor description, #cores, #threads, and 32/64 bit ISA.
- Memory description, memory mapped IO region and other regions.
- Devices supported and their specifications.

Describes Software

- Build/Execute parameters
- User function to hardware thread mapping.
- Specify dynamic memory pool
- Program traps/interrupts.

config.yaml - describes project

It only has two components.

```
Hardware:

   . . .


Software:

   . . .
```

config.yaml - Hardware

Describes the Ajit specific hardware configuration to CoRTOS.

```
Hardware:
    Processor: …
    Memory: …
    Devices: …
```

config.yaml - Hardware.Processor

```
Processor:
    Cores: 1              # 1 to 8 only
    ThreadsPerCore: 2     # 1 or 2 only
    ISA: 32               # 32/64 bit (Default: 32)
```

config.yaml - Hardware.Memory

Describes various hardware memory regions. E.g. Flash Memory, RAM, NC RAM, Memory Mapped IO

NC RAM is a special non-cacheable ram region in the hardware memory system. It is for the devices to directly write to memory without caring about the processor cache.

```
Memory:
  MaxPhysicalAddrBitWidth: 36   # hw bus addr bit width
  Flash:
    StartAddr: 0x0          # physical address
    SizeInMegaBytes: 16
    Permissions: RXC        # (Read,Write,eXecute,Cacheable)
  RAM:
    StartAddr: 0x40000000  # physical address
    SizeInMegaBytes: 128
    Permissions: RWXC
  NCRAM:
    StartAddr: 0x8000000   # physical address
    SizeInMegaBytes: 16
    Permissions: RW
  MMIO: # Memory Mapped IO
    StartAddr: 0xFFFF0000  # physical address
    EndAddr:   0xFFFFFFFF
    Permissions: RW
```

config.yaml - Hardware.Devices

./AjitPublicResources/tools/ajit_access_routines_mt/include/device_addresses.h

All devices communicate with the processor using memory mapped IO.

```
Devices:
    - Name: Timer
      MemoryRegion:
        StartAddr: 0xFFFF3100 # physical addr (in MMIO)
        SizeInBytes: 256 # 16x16
        Permissions: RW
      NamedRegisters:
        Control: 0xFFFF3100
    - Name: InterruptController
        …
    - Name: Serial
      MemoryRegion:
        …
    - Name: ScratchArea
      MemoryRegion:
        StartAddr: 0xFFFF2C00
        SizeInBytes: 1024
        Permissions: RW
```

config.yaml - Software

```
Software:
    StartupFuncName: …
    BuildAndExecute: …
    ProgramThreads: …
    DynamicMemroy: …
    Interrupts: …
    Traps: …
```

config.yaml - Software.StartupFuncName

A startup function with “void (void)” signature.

```
StartupFuncName: user_func
```

user_func() is invoked on thread (0,0) after system initialization by CoRTOS is complete. All other user functions on all threads start after this function completes.

config.yaml - Software.BuildAndExecute

Specifies the build and runtime parameters (for Ajit's C Reference Model) of the project.

The **BuildArgs** parameter is passed to the `ajit_C_reference_model`. Refer the program to see all its options.

Note: Source sub-folders in the project can be specified in BuildArgs using `-C` and `-S` options.

```
BuildAndExecute:
    # Optional: Specify the log level. Default is NONE.
    # ALL<TRACE<DEBUG<INFO<ERROR<CRITICAL<NONE
    LogLevel: DEBUG
    OptimizationLevel: 2 # i.e. O2 (0, 1 or 2)
    Debug: No # Yes/No (Default: No)
    FirstDebugPort: 8888
    BuildArgs: "-D CORTOS_ENV"
```

Or for debug build use: cortos build -g

config.yaml - Software.ProgramThreads

Maps user functions to hardware threads.

```
ProgramThreads:
    - CortosInitCalls:
        - func01    # no naming restrictions
        - func02
      StackSize:
        SizeInKiloBytes: 16
    - CortosInitCalls:
        - func10    # no naming restrictions
      StackSize:
        SizeInBytes: 2048
```

func01 and func02 are execute one after the other on thread (0,0) and func10 is executed on the next available hardware thread.

config.yaml - Software.DynamicMemory

Size of dynamic memory pool can be specified.

```
DynamicMemory: # Dynamic Memory Configuration.
  SizeInKiloBytes: 1000    # default is 4KB
```

When DynamicMemory is not specified, the system assumes a default size of 4KB.

config.yaml - Software.Interrupts

User can handle some hardware interrupts.

```
#########################################################
  # Hardware interrupts.
  # Specify a list of hardware interrupt handlers
  # Index: An index from 1 to 15.
  # Function: <func_name>  (of type void (*func_name)())

#########################################################
  Interrupts: # hardware interrupts
    - Index: 12  # serial interrupt index
      Function: my_serial_interrupt_handler
    - Index: 15
      Function: int_15_handler
```

config.yaml - Software.Traps

User can handle software traps. (Note the function signature type below)

```
#########################################################
  # Software traps
  # Specify a list of hardware interrupt handlers
  # Index: An index from 1 to 15.
  # Function: <func_name>
  #           void (*func_name)(uint32_t,uint32_t,uint32_t)
#########################################################
Traps: # software traps
  - Index: 12  # trap index
    Function: trap_12_handler     # no naming restrictions
  - Index: 15
    Function: trap_15_handler
```

CoRTOS API

CoRTOS API (cortos.h)

This presentation may get out-dated when APIs are updated. The true source of APIs are the header files provided by CoRTOS. The APIs are well documented there.

CoRTOS environment:

#define **CORTOS_ENV**

is available to all projects

Locks and Synchronization (cortos_locks.h)
Get a pointer to a lock and then use it.

Queue and Message Passing (cortos_queues.h)
Dynamically/Statically allocate a queue and start using it.

Dynamic Memory (cortos_bget.h)
Dynamically allocate memory at runtime.

Utilities: Printing Output etc. (cortos_utils.h)
cortos_printf(), cortos_exit() etc.

Logging: Printing log messages. (cortos_logging.h)
CORTOS_DEBUG(), CORTOS_INFO(), …

***Examples in CoRTOS are a good demonstration of these APIs.**

API - using Locks (cortos_locks.h)

CoRTOS provides a pool of 256 cacheable and 256 non-cacheable (NC) locks.

```
uint8_t* lockPtr = cortos_reserveLock(1 /*i.e. NC lock*/);

cortos_lock_acquire_buzy(lockPtr);

// critical section code

cortos_lock_release(lockPtr);


//free the lock variable
cortos_freeLock(lockPtr);
// after free, the value in lockPtr should not be used.
```

See example_100

API - using Queues (cortos_queues.h)

User can dynamically/statically allocate queues. Here we show dynamic allocation.

writeCount and readCount hold the actual number of messages written to or read from the queue.

See example_150

```
struct CortosQueueHeader * volatile hdr = 0;
uint32_t msgs_a[4], msgs_b[4];

hdr = cortos_reserveQueue(
      sizeof(uint32_t) /*single msg size in bytes*/,
      2 /*length i.e. max messages in the queue*/,
      1 /*1 means non-cacheable*/);

writeCount = cortos_writeMessages(
            hdr, /* pointer to the queue */
            (uint8_t*)(msgs_a), /* message source */
            4 /*max number of messages to be written*/);

readCount = cortos_readMessages(
            hdr,
            (uint8_t*)(msgs_b),/* message destination */
            4  /*max number of messages to be read*/);
```

API - dynamic memory (cortos_bget.h)

Dynamically allocate and deallocate memory at runtime.

`_ncram` is for non-cacheable RAM area where the memory is allocated.

See

- example_200
- example_210

```
uint32_t * volatile arr = 0;
arr = (uint32_t*)cortos_bget(sizeof(uint32_t)*20);

// use the memory

cortos_brel(arr); // free memory
```

```
uint32_t * volatile arr = 0;
arr =
(uint32_t*)cortos_bget_ncram(sizeof(uint32_t)*20);

// use the memory

cortos_brel_ncram(arr); // free memory
```

API - utilities (cortos_utils.h)

Some utilities provided by cortos.

These can be seen in almost all the examples.

```
cortos_printf("hello %d", value); // thread safe

cortos_exit(exit_integer_code);
```

Ajit's access routines provide many more APIs.

API - logging (cortos_logging.h)

Thread safe logging support by CoRTOS. **Use these instead of `cortos_printf`.**

These can be seen in almost all the examples.

```
CORTOS_DEBUG("hello %d", value);

CORTOS_INFO("hello %d", value);
```

Along with the user message, this prints the thread id, source line number, and the number of elapsed ticks of the processor.

Logging Levels: ALL < TRACE < DEBUG < INFO < ERROR < CRITICAL < NONE

Example, if logging level is set to INFO, then TRACE and DEBUG messages disappear.

config.yaml: **Software.BuildAndExecute.LogLevel**: INFO

API - interrupts/traps (cortos_traps.h)

User can program 15 hardware interrupts and 15 software interrupts.

Invoke a software trap from C code:

```
__AJIT_SW_TRAP(5);  // ta 5
```

Please see the following examples:

- **example_310 (interrupts)**
- **example_320 (traps)**

Declare volatiles carefully!

Global variables used across threads may need to be declared volatile by the programmer.

Please note the difference below:

```
volatile int * ptr; // a volatile int value

int * volatile ptr; // a volatile pointer

// a volatile pointer to a volatile int value
volatile int * volatile ptr;
```

CoRTOS Examples Directory

CoRTOS Examples Directory

It contains various `example_XXX` directories. Go into any of these directories to build and run them

```
cd example_150

./build.sh     # builds the example

./run.sh       # run using C Model
```

Remember to set paths of the AJIT's setup.

Example have config.yaml files which are meant to be used as reference for user projects.

LOCATION: In the ajit-toolchain repository: cd os/rtos/cortos2/examples;

Thank You

TODOs

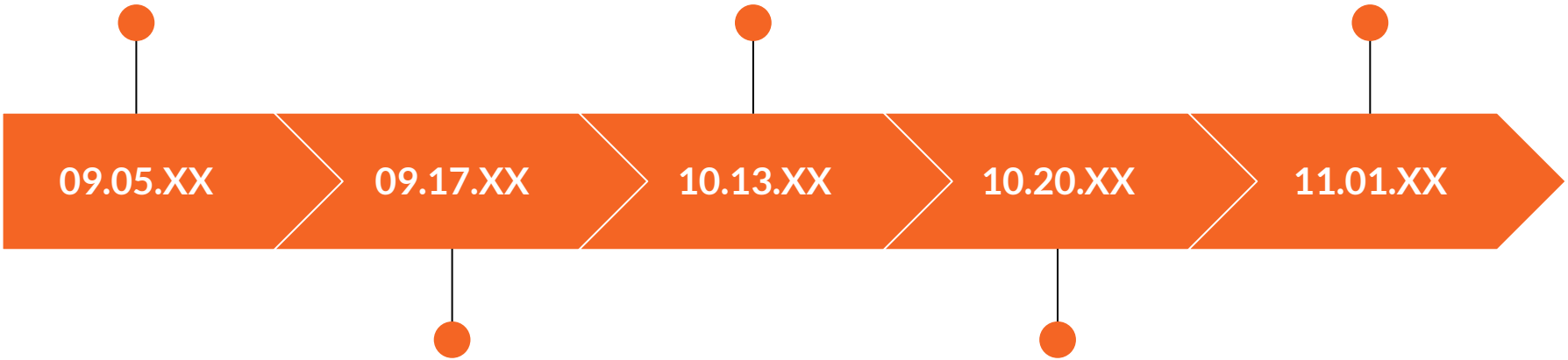
1. Standardise the API (done).
2. Improve system feedback (improved).
3. Add more tests (added three tests).

Questions/Suggestions?

Lorem ipsum dolor sit amet, consectetur adipiscing elit

Lorem ipsum dolor sit amet, consectetur adipiscing elit

Lorem ipsum dolor sit amet, consectetur adipiscing elit



Lorem ipsum dolor sit amet, consectetur adipiscing elit

Lorem ipsum dolor sit amet, consectetur adipiscing elit