AVDS Platform - Xenomai RT

Vehicle design and simulation system

Exported on 03/29/2021

Table of Contents

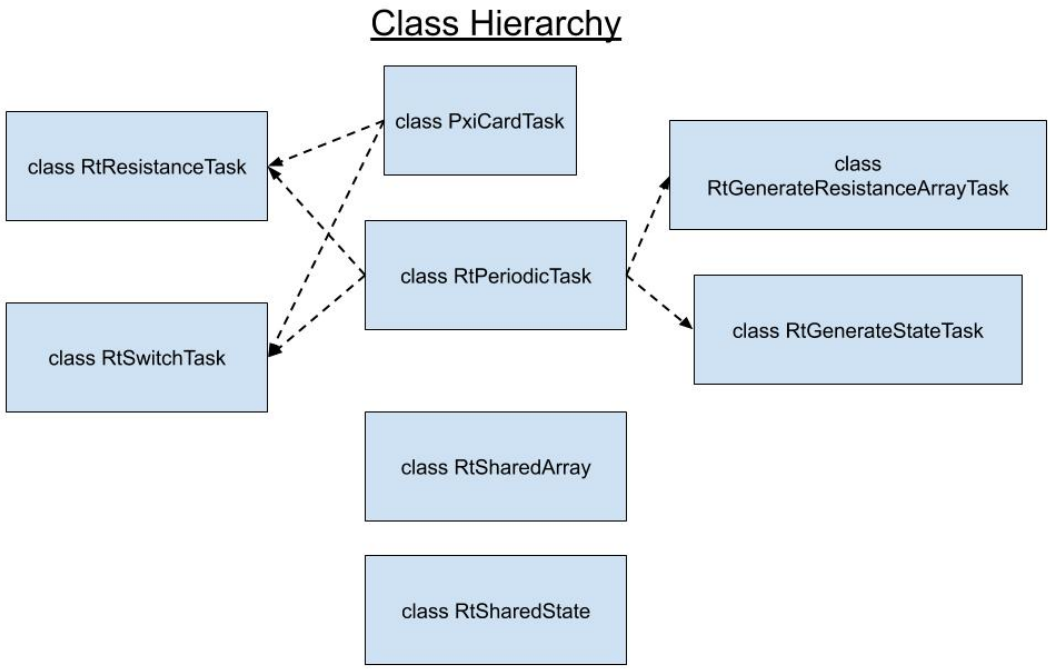
1 Redis

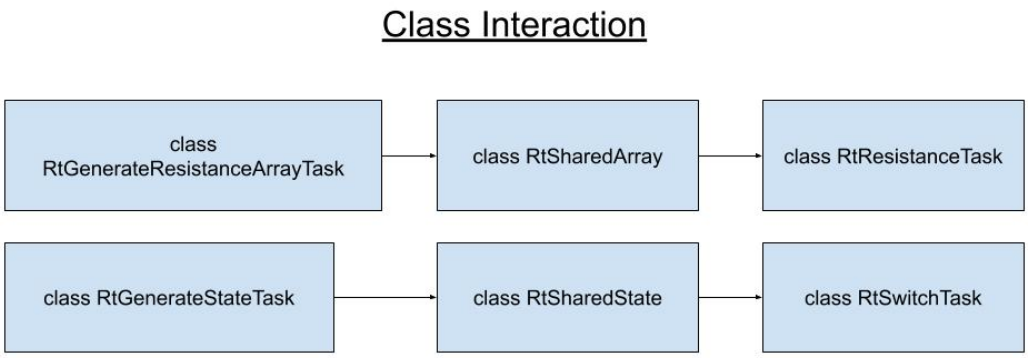
2020-03-05 Work Log - Learning Redis[1]

1 https://docs.google.com/document/d/1RI4-OUfQd6AkXbEJoErlnjT-3JiqNgDQzYPxUbtDs1U/edit?usp=sharing

2 From Everything to Xenomai RT Task and Beyond

2.1 Class Hierarchy





Class Hierarchy and Class Interaction at commit 63b4d9f3418[2]

1. class PxiCardTask
    a. Mainly for storing PXI interface card related variables and opening a PXI card
    b. OpenCard() will open a PXI interface card with a card num provided
        i. simplified function requiring only cardNum as argument to open a card
    c. ViewAllSubunits() will show all subunits within a card
        i. simplified function to view all subunits within a card
    d. PxiCardTask's constructor will not access the PXI interface card

2 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits/63b4d9f3418d4b7d190abc27a2fc1c4f4cb9c512

PxiCardTask.h

```
#ifndef _PXICARDTASK_H_
#define _PXICARDTASK_H_

#include <stdio.h>

#include <Pilpxi.h>

class PxiCardTask
{
public:
  static DWORD mBit;
  static DWORD mBus;
  static DWORD mBuses[100];
  static DWORD mCardNum;
  static DWORD mData[100];
  static DWORD mDevices[100];
  static DWORD mDevice;
  static DWORD mNumInputSubunits;
  static DWORD mNumOfFreeCards;
  static DWORD mNumOutputSubunits;
  static DWORD mResistance;
  static DWORD mResistances[100];
  static DWORD mSubunit;

  static CHAR mCardId[100];

  static BOOL mState;

public:
  PxiCardTask();
  void OpenCard(DWORD cardNum);
  void ViewAllSubunits(DWORD cardNum);
};

#endif // _PXICARDTASK_H_
```

PxiCardTask.cpp

```
#include <PxiCardTask.h>

DWORD PxiCardTask::mBit;
DWORD PxiCardTask::mBus;
DWORD PxiCardTask::mBuses[100];
DWORD PxiCardTask::mCardNum;
DWORD PxiCardTask::mData[100];
DWORD PxiCardTask::mDevices[100];
DWORD PxiCardTask::mDevice;
DWORD PxiCardTask::mNumInputSubunits;
DWORD PxiCardTask::mNumOfFreeCards;
DWORD PxiCardTask::mNumOutputSubunits;
DWORD PxiCardTask::mResistance;
DWORD PxiCardTask::mResistances[100];
DWORD PxiCardTask::mSubunit;

CHAR PxiCardTask::mCardId[100];

BOOL PxiCardTask::mState;

PxiCardTask::PxiCardTask()
{}

void PxiCardTask::OpenCard(DWORD cardNum)
{
  PIL_CountFreeCards(&mNumOfFreeCards);
  PIL_FindFreeCards(mNumOfFreeCards, mBuses, mDevices);
  PIL_OpenSpecifiedCard(mBuses[cardNum-1], mDevices[cardNum-1], &mCardNum);
  mBus = mBuses[mCardNum-1];
  mDevice = mDevices[mCardNum-1];
  PIL_CardId(mCardNum, mCardId);
  PIL_EnumerateSubs(mCardNum, &mNumInputSubunits, &mNumOutputSubunits);

  printf("Opening cardNum: %d, mCardNum: %d, bus: %d, device: %d, card id: %s, "
    "# input subunits: %d, # output subunits: %d\n",
    cardNum, mCardNum, mBus, mDevice, mCardId, mNumInputSubunits,
mNumOutputSubunits);
}

void PxiCardTask::ViewAllSubunits(DWORD cardNum)
{
  PIL_EnumerateSubs(mCardNum, &mNumInputSubunits, &mNumOutputSubunits);
  for(auto i{0u}; i < mNumOutputSubunits; ++i)
  {
    BOOL out;
    CHAR subType[100];
    DWORD data[100];
    PIL_SubType(cardNum, i, out, subType);
    PIL_ViewSub(cardNum, i, data);

    printf("Subunit #%d (%s) = %d Ohm", i, subType, data[0]);
  }
```

```
52  }
```

2. class RtPeriodicTask
    a. class for storing Xenomai RT related variables with the constructor
    b. no functionality, it's can be treated as a virtual class
    c. most of the functions are deleted, designed to be defeind when inherited by other classes
        i. any task inheriting from it will need to define "int StartRoutine()" and "void Routine(void*)"
    d.

RtPeriodicTask.h

```
1   #ifndef _RTPERIODICTASK_H_
2   #define _RTPERIODICTASK_H_
3
4   #include <alchemy/task.h>
5
6   class RtPeriodicTask
7   {
8   public:
9     RT_TASK mRtTask;
10
11    static RTIME mNow;
12    static RTIME mOneSecondTimer;
13    static RTIME mPrevious;
14
15    const char* mName;
16
17    int mCoreId;
18    int mMode;
19    int mPeriod;
20    int mPriority;
21    int mStackSize;
22
23    cpu_set_t mCpuSet;
24
25  public:
26    RtPeriodicTask() = delete;
27    RtPeriodicTask(
28      const char* name, const int stackSize, const int priority, const int mode,
29      const int period, const int coreId);
30    int StartRoutine() = delete;
31    static void Routine(void*) = delete;
32  };
33
34  #endif // _RTPERIODICTASK_H_
```

RtPeriodicTask.cpp

```cpp
#include <RtPeriodicTask.h>

RTIME RtPeriodicTask::mNow;
RTIME RtPeriodicTask::mOneSecondTimer;
RTIME RtPeriodicTask::mPrevious;

RtPeriodicTask::RtPeriodicTask(
  const char* name, const int stackSize, const int priority, const int mode,
  const int period, const int coreId)
{
  mName = name;
  mStackSize = stackSize;
  mPriority = priority;
  mMode = mode;
  mPeriod = period;
  mCoreId = coreId;

  CPU_ZERO(&mCpuSet);
  CPU_SET(mCoreId, &mCpuSet);
}
```

3. class RtSharedArray
    a. Shared Array of DWORD to store values of resistance to be written to the PXI resistance card
    b. includes a Xenomai RT_MUTEX for thread safety.
        i. Unless specified at construction, the default acquiring timeout=TM_INFINITE

C. **RtSharedArray.h**

```
#ifndef _RTSHAREDARRAY_H_
#define _RTSHAREDARRAY_H_

#include <vector>

#include <alchemy/mutex.h>

#include <Pilpxi.h>

class RtSharedArray
{
public:
  DWORD mArray[100];
  RT_MUTEX mMutex;
  RTIME mTimeout;
  const char* mName;

public:
  RtSharedArray() = delete;
  RtSharedArray(const char* name, const RTIME &timeout=TM_INFINITE);
  void Set(unsigned int index, DWORD element);
  void SetArray(const std::vector<DWORD> &elements);
  DWORD Get(unsigned int index);
};

#endif // _RTSHAREDARRAY_H_
```

RtSharedArray.cpp

```cpp
#include <RtSharedArray.h>

RtSharedArray::RtSharedArray(const char* name, const RTIME &timeout)
  : mName(name)
  , mTimeout(timeout)
{
  rt_mutex_create(&mMutex, mName);
}

void RtSharedArray::Set(unsigned int index, DWORD element)
{
  rt_mutex_acquire_until(&mMutex, mTimeout);
  mArray[index] = element;
  rt_mutex_release(&mMutex);
}

void RtSharedArray::SetArray(const std::vector<DWORD> &elements)
{
  rt_mutex_acquire_until(&mMutex, mTimeout);
  for(auto i{0u}; i < elements.size(); ++i)
  {
    mArray[i] = elements[i];
  }
  rt_mutex_release(&mMutex);
}

DWORD RtSharedArray::Get(unsigned int index)
{
  rt_mutex_acquire_until(&mMutex, mTimeout);
  auto element = mArray[index];
  rt_mutex_release(&mMutex);
  return element;
}
```

4. class RtSharedState
    a. Shared State (BOOL) to store a state to be written to a PXI switching card
    b. includes a Xenomai RT_MUTEX for thread safety.
        i. Unless specified at construction, the default acquiring timeout=TM_INFINITE

c. **RtSharedState.h**

```
#ifndef _RTSHAREDSTATE_H_
#define _RTSHAREDSTATE_H_

#include <alchemy/mutex.h>

#include <Pilpxi.h>

class RtSharedState
{
private:
  BOOL mState;
  RT_MUTEX mMutex;
  RTIME mTimeout;
  const char* mName;

public:
  RtSharedState() = delete;
  RtSharedState(const char* name, const RTIME &timeout=TM_INFINITE);
  void Set(bool b);
  BOOL Get();
};

#endif // _RTSHAREDSTATE_H_
```

RtSharedState.cpp

```
#include <RtSharedState.h>

RtSharedState::RtSharedState(const char* name, const RTIME &timeout)
  : mName(name)
  , mTimeout(timeout)
{
  rt_mutex_create(&mMutex, mName);
}

void RtSharedState::Set(bool b)
{
  rt_mutex_acquire_until(&mMutex, mTimeout);
  mState = b;
  rt_mutex_release(&mMutex);
}

BOOL RtSharedState::Get()
{
  rt_mutex_acquire_until(&mMutex, mTimeout);
  auto currentState = mState;
  rt_mutex_release(&mMutex);
  return currentState;
}
```

5. class RtGenerateResistanceArrayTask

a. contains a Simulink Model
    i. this member variable is added after imported Simulink Generated C++ code

```
#include <testing.h>

static testingModelClass mModel;
```

b. inherits RtPeriodicTask
    i. because it would continuously run step() function from the Simulink Model

```
#include <RtPeriodicTask.h>

class RtGenerateResistanceArrayTask : public RtPeriodicTask
```

c. contains a RtSharedArray to pass the resistance values to another RT Task responsible for accessing the PXI card
    i. multiple RT Tasks would need some way to share information, with thread safety implemented

```
#include <RtSharedArray.h>

static std::shared_ptr<RtSharedArray> mRtSharedArray;
```

d. constructor will have to include all arguments needed to create a RtPeriodicTask
    i.

RtGenerateResistanceArrayTask.h

```
RtGenerateResistanceArrayTask(
  const char* name, const int stackSize, const int priority, const int mode,
  const int period, const int coreId=0);
```

RtPeriodicTask.h

```
RtPeriodicTask(
  const char* name, const int stackSize, const int priority, const int mode,
  const int period, const int coreId);
```

e. implements RtPeriodicTask's StartRoutine() and Routine() functions

i. **RtGenerateResistanceArrayTask.cpp**

```
11  int RtGenerateResistanceArrayTask::StartRoutine()
12  {
13    mlockall(MCL_CURRENT|MCL_FUTURE);
14
15    int e1 = rt_task_create(&mRtTask, mName, mStackSize, mPriority, mMode);
16    int e2 = rt_task_set_periodic(&mRtTask, TM_NOW,
    rt_timer_ns2ticks(mPeriod));
17    int e3 = (mCoreId > 0) ? rt_task_set_affinity(&mRtTask, &mCpuSet) : 0;
18    int e4 = rt_task_start(&mRtTask, &Routine, NULL);
19
20    if(e1 | e2 | e3 | e4)
21    {
22      printf("Error with RtGenerateResistanceArrayTask::StartRoutine().
    Exiting.\n");
23      exit(-1);
24    }
25  }
26
27  void RtGenerateResistanceArrayTask::Routine(void*)
28  {
29    mModel.initialize();
30
31    std::vector<DWORD> resistances;
32
33    while(true)
34    {
35      resistances.clear();
36
37      // TODO: make 10u a size for share array
38      for(auto i{0u}; i < 10u; ++i)
39      {
40        mModel.step();
41        DWORD subunit = i;
42        DWORD resistance = mModel.testing_Y.Out1 * 2 + 10;
43
44        resistances.push_back(resistance);
45      }
46      mRtSharedArray->SetArray(resistances);
47
48      rt_task_wait_period(NULL);
49    }
50  }
```

6. class RtGenerateStateTask
    a. Same as RtGenerateResistanceArrayTask except that it generates a state to be written to RtSharedState
7. class RtResistanceTask
    a. Gets the values of resistance from RtGenerateResistanceArrayTask, which would have Simulink model step(), and then access a PXI interface card to change resistance values accordingly
    b. inherits RtPeriodicTask because it will be receiving resistances to update periodically from RtGenerateResistanceArrayTask
    c. inherits PxiCardTask because it will access the PXI interface card

```
class RtResistanceTask: public RtPeriodicTask, public PxiCardTask
```

d. has RtSharedArray to receive resistance values

```
    auto previousResistance = mData[0];
    mData[0] = mRtSharedArray->Get(i);
```

e. implements RtPeriodicTask's StartRoutine() and Routine() functions

RtResistanceTask.cpp

```
int RtResistanceTask::StartRoutine()
{
  mlockall(MCL_CURRENT|MCL_FUTURE);

  int e1 = rt_task_create(&mRtTask, mName, mStackSize, mPriority, mMode);
  int e2 = rt_task_set_periodic(&mRtTask, TM_NOW, rt_timer_ns2ticks(mPeriod));
  int e3 = (mCoreId > 0) ? rt_task_set_affinity(&mRtTask, &mCpuSet) : 0;
  int e4 = rt_task_start(&mRtTask, &Routine, NULL);

   if(e1 | e2 | e3 | e4)
   {
     printf("Error launching periodic task SetSubunitResistanceRoutine. Exiting.
\n");
     return -1;
   }
}

void RtResistanceTask::Routine(void*)
{
  printf("Accessing bus %d, device %d, target resistance %d\n", mBus, mDevice,
mResistance);
  mPrevious = rt_timer_read();
  while(true)
  {
    for(auto i{0u}; i < mNumOutputSubunits; ++i)
    {
      PIL_ViewSub(mCardNum, i, mData);
      auto previousResistance = mData[0];
      mData[0] = mRtSharedArray->Get(i);
      PIL_WriteSub(mCardNum, i, mData);

      rt_task_wait_period(NULL);

      mNow = rt_timer_read();

      if(static_cast<long>(mNow - mOneSecondTimer) / RtMacro::kNanosecondsToSeconds
> 0)
      {
        printf("Time elapsed for task: %ld.%ld microseconds\n",
          static_cast<long>(mNow - mPrevious) / RtMacro::kNanosecondsToMicroseconds,
          static_cast<long>(mNow - mPrevious) %
RtMacro::kNanosecondsToMicroseconds);
        mOneSecondTimer = mNow;

        /* show all subunits */
        PIL_EnumerateSubs(mCardNum, &mNumInputSubunits, &mNumOutputSubunits);
        for(auto i{0u}; i < mNumOutputSubunits; ++i)
        {
          BOOL out;
          CHAR subType[100];
          DWORD data[100];
          PIL_SubType(mCardNum, i, out, subType);
          PIL_ViewSub(mCardNum, i, data);
```

```
50            printf("Subunit #%d (%s) = %d Ohm\n", i, subType, data[0]);
51          }
52          printf("\n");
53        }
54
55        mPrevious = mNow;
56      }
57    }
58  }
```

8. class RtSwitchTask
    a. the same as RtResistanceTask except that it's changing the state of a switch in a PXI switching card

RtSwitchTask.h

```
1  #ifndef _RTSWITCHTASK_H_
2  #define _RTSWITCHTASK_H_
3
4  #include <sys/mman.h>
5
6  #include <memory>
7
8  #include <RtMacro.h>
9  #include <RtPeriodicTask.h>
10 #include <RtSharedState.h>
11
12 #include <PxiCardTask.h>
13
14 class RtSwitchTask: public RtPeriodicTask, public PxiCardTask
15 {
16 public:
17   static std::shared_ptr<RtSharedState> mRtSharedState;
18
19 public:
20   RtSwitchTask(
21     const char* name, const int stackSize, const int priority, const int mode,
22     const int period, const int coreId=0);
23   int StartRoutine();
24   static void Routine(void*);
25   ~RtSwitchTask();
26 };
27
28 #endif // _RTSWITCHTASK_H_
```

RtSwitchTask.cpp

```
#include <RtSwitchTask.h>

std::shared_ptr<RtSharedState> RtSwitchTask::mRtSharedState;

RtSwitchTask::RtSwitchTask(
  const char* name, const int stackSize, const int priority, const int mode,
  const int period, const int coreId)
  : RtPeriodicTask(name, stackSize, priority, mode, period, coreId)
{}

int RtSwitchTask::StartRoutine()
{
  mlockall(MCL_CURRENT|MCL_FUTURE);

  int e1 = rt_task_create(&mRtTask, mName, mStackSize, mPriority, mMode);
  int e2 = rt_task_set_periodic(&mRtTask, TM_NOW, rt_timer_ns2ticks(mPeriod));
  int e3 = (mCoreId > 0) ? rt_task_set_affinity(&mRtTask, &mCpuSet) : 0;
  int e4 = rt_task_start(&mRtTask, &Routine, NULL);

  if(e1 | e2 | e3 | e4)
  {
    printf("Error launching periodic task SetSubunitSwitchState. Exiting.\n");
    return -1;
  }
  printf("RtSwitchTask running on CoreId: %d\n", mCoreId);
}

void RtSwitchTask::Routine(void*)
{
  printf("Accessing bus %d, device %d, target resistance %d\n", mBus, mDevice,
mResistance);
  mPrevious = rt_timer_read();
  BOOL prevState;
  while(true)
  {
    // TODO: check how many bits are in the subunit
    PIL_ViewBit(mCardNum, mSubunit, mBit, &prevState);

    auto setState = mRtSharedState->Get();

    PIL_OpBit(mCardNum, mSubunit, mBit, setState);
    PIL_ViewBit(mCardNum, mSubunit, mBit, &mState);

    rt_task_wait_period(NULL);

    mNow = rt_timer_read();

    if(static_cast<long>(mNow - mOneSecondTimer) / RtMacro::kNanosecondsToSeconds >
0)
    {
      printf("Time elapsed for task: %ld.%ld microseconds\n",
        static_cast<long>(mNow - mPrevious) / RtMacro::kNanosecondsToMicroseconds,
        static_cast<long>(mNow - mPrevious) % RtMacro::kNanosecondsToMicroseconds);
```

```
52
53        printf("State changed from %s -> %s (setState = %s)\n",
54          prevState ? "true" : "false", mState ? "true" : "false", setState ? "true" :
"false");
55
56        mOneSecondTimer = mNow;
57      }
58      mPrevious = mNow;
59    }
60  }
61
62  RtSwitchTask::~RtSwitchTask()
63  {
64    int e = rt_task_delete(&mRtTask);
65    if(e)
66    {
67      printf("Error deleting RtSwitchTask::mRtTask. Exiting.\n");
68      exit(-1);
69    }
70    printf("RtSwitchTask::mRtTask deleted.\n");
71  }
```

2.2 Communication Between Xenomai Tasks via Message Queue

Scenario: When two programs each will start a Xenomai task and communicate via message queue

1. Exist in the same session
    a. ./message_queue_receive --session=send-receive
    b. ./message_queue_send --session=send-receive
2. One of the two would create a RT_QUEUE
    a. rt_queue_create()
3. The other one would bind the RT_QUEUE
    a. rt_bind_queue()
4. Starts communication

Reference:

1. [Xenomai] xenomai-3.0-rc5 : binding named semaphores from external process[3]
2. [Xenomai] rt_taks_bind() returns -EAGAIN for existing task[4]

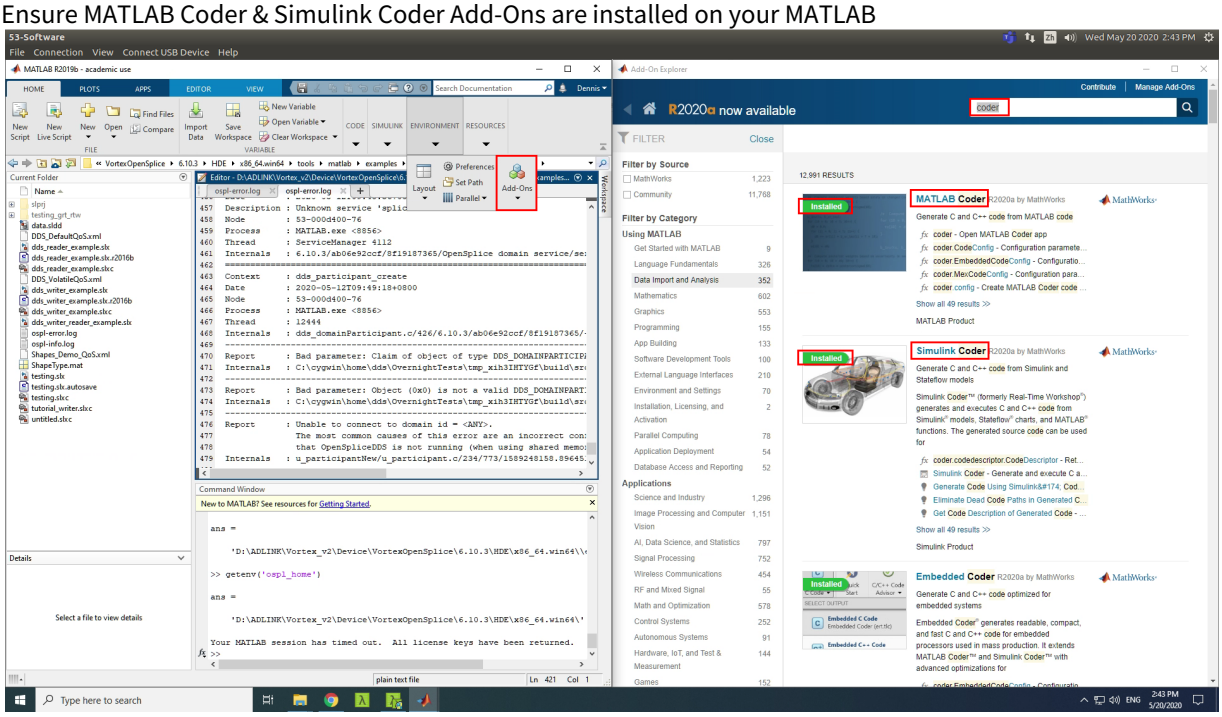
2.3 From Windows Simulink to Ubuntu CMake from MATLAB Simulink

Having to run Simulink Model within Xenomai RT (Real-Time) environment would require some steps, follow the steps below to achieve doing so:

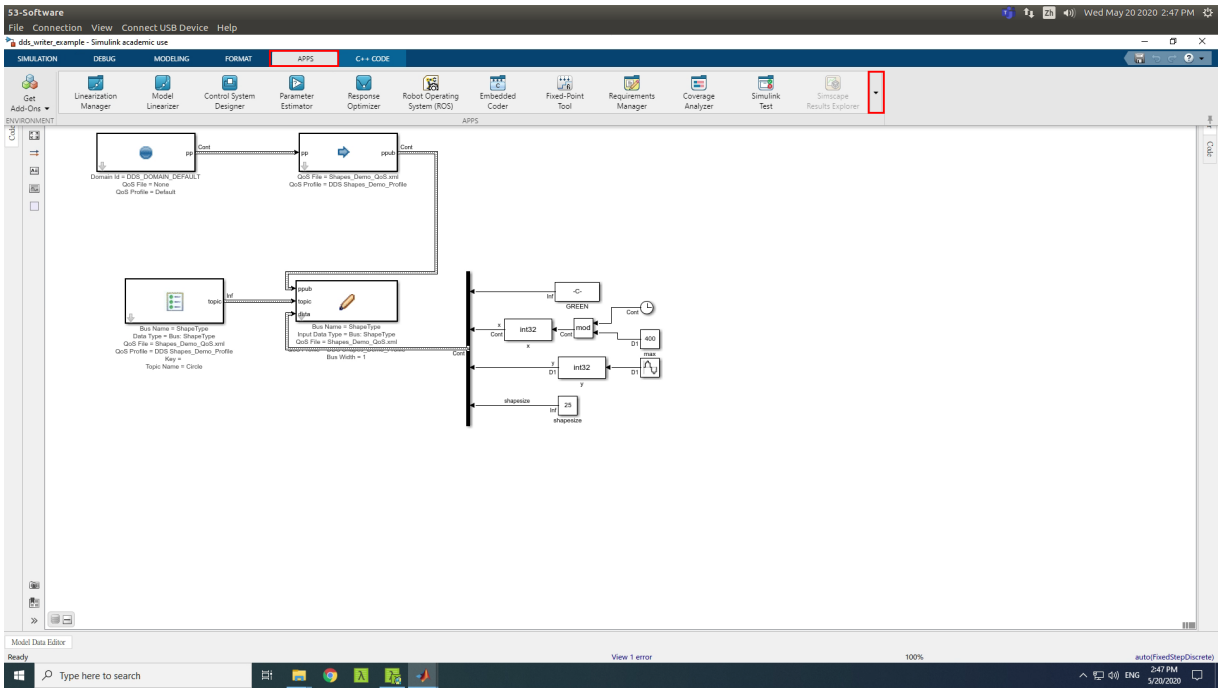
1. Generate C++/C Code from Simulink

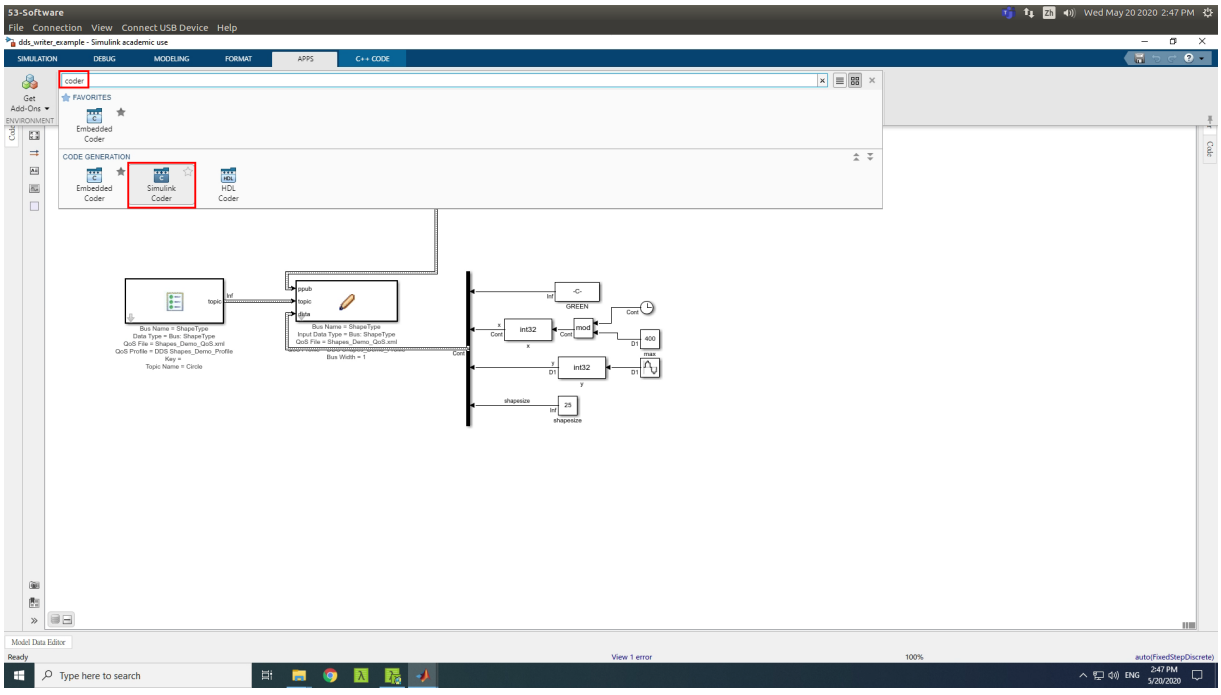
3 https://xenomai.org/pipermail/xenomai/2015-July/034665.html
4 https://xenomai.org/pipermail/xenomai/2017-June/037395.html

1. Ensure MATLAB Coder & Simulink Coder Add-Ons are installed on your MATLAB



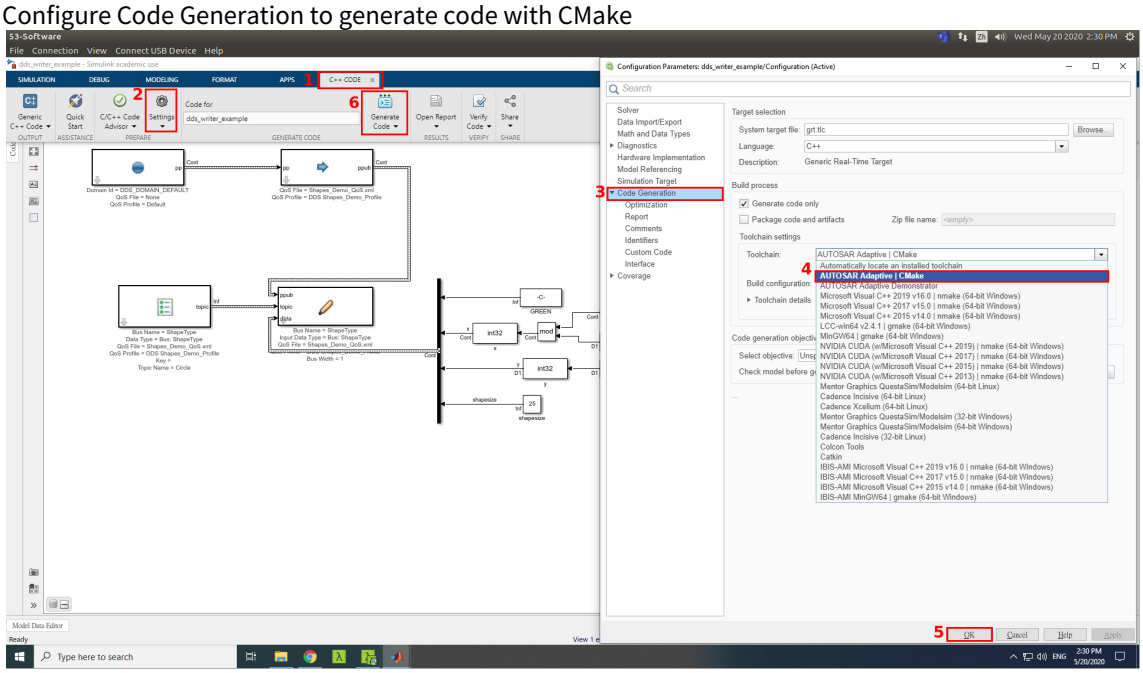
2. Enable Simulink Coder





3. Generate C++/C Code with Simulink Coder

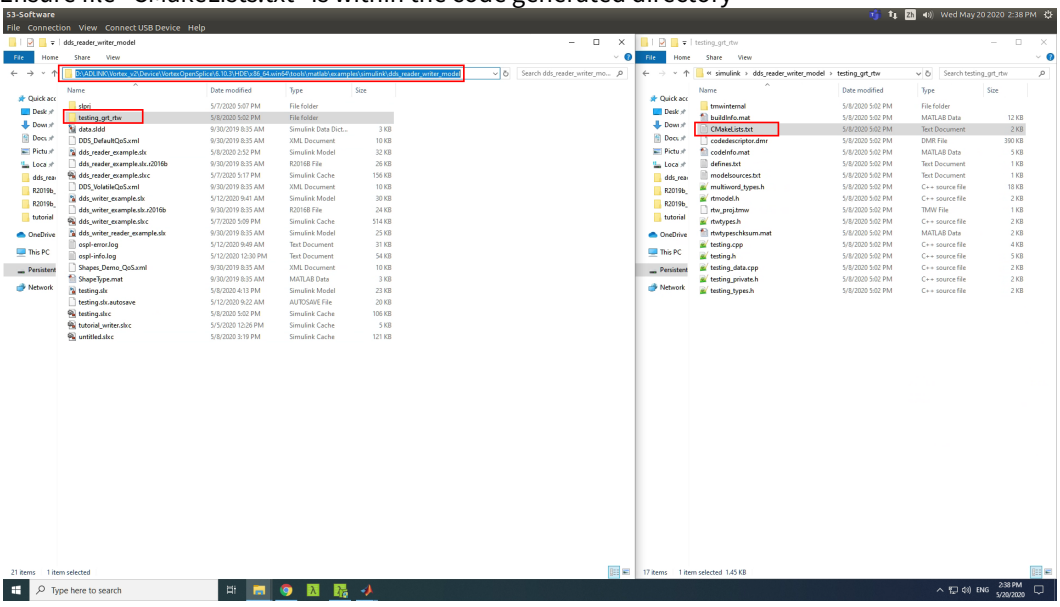
a. Configure Code Generation to generate code with CMake



b. Verify Successful Code Generation

i. in the same path of your Simulink Model, a folder should be created with the format of [model_name]_grt_rtw

ii. Ensure file “CMakeLists.txt” is within the code generated directory



- Compile Simulink Generated Code on Xenomai Enabled Machine
  a. Create a CMake Project
     i. Manually

1. Physically copy the generated code to Xenomai machine (see reposiroty rt-hil-simulation commit 0958e53001c[5], folder "simulink_generated_code/testing_grt_rtw/")
2. Copy required MATLAB directory/files for compilation:
    a. [MATLAB_install_directory]/extern/include/
    b. [MATLAB_install_directory]/simulink/include/
    c. [MATLAB_install_directory]/rtw/c/
    d. And any other directory your Simulink Model depends on, if application.
    e. see src/matlab_code[6]
3. Create a new CMakeLists.txt (use the generated CMakeLists.txt as template)
    a. In the new CMakeLists.txt add the following
        i. add_executable()
        ii. target_include_directories()
        iii. A working CMakeLists.txt can be observed in this commit CMakeLists.txt[7]
            i. Omit the lines starting with #
            ii. Observe the added lines shown in green
        iv. Note: the location of the CMakeLists.txt will affect how you specify the paths for each cmake commands within the CMakeLists.txt file
        v. Note: for more cmake related information, check the working CMakeLists.txt[8] and the cmake folder[9]
4. Delete the old CMakeLists.txt from Simulink Code Generation
    a. old CMakeLists.txt to be deleted[10]
    b. Note: the point of Simulink generating code with CMakeLists.txt is so that we can have a template CMakeLists.txt and also have the project already structured for CMake.

ii. Automatically
1. clone rt-hil-simulation[11]
2. locate python script file named config_cmake.py[12] (see CMake Project Config and File Collector(see page 27) for details)
3. run

```
python.exe config_cmake.py [path_to_cmake_proj]
```

    a. A folder with new CMakeLists.txt and necessary files should be created in the same directory

b. Write a simple C++/C program that includes the Simulink Model
    i. see commit 2f6520c2b62[13]
    ii. Create a file within the Simulink Generated Code directory

---

5 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/dds-motor-controller/commits/0958e53001c2b1401b1ada11aedb026f201d6e9a
6 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/matlab_code
7 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/commits/62381735210feafd1eeb80834937ca2243ab77dd
8 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/browse/src/motor_control_unit/CMakeLists.txt
9 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake
10 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake
11 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse
12 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/python_code/config_cmake.py
13 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/commits/cf28e2c379efe16336ba373233ec90912f81b7ed

1. e.g. `touch testing_grt_rtw/testing_main.cpp`

iii. include header file
   1. "#include <[simulink_model_name].h>"
   2. e.g. "#include <testing.h>"

iv. Use your model within your int main()
   1. instantiate your model
      a. e.g. "auto testingModel = testingModelClass()"
      b. the specific naming of the class name can be found in file "[simulink_model_name].h"
         i. src/simulink_generated_code/switching_testing_grt_rtw/testing.h[14]
   2. initialize the model
      a. e.g. "testingModel.initialize()"
   3. have the model step once
      a. e.g. "testingModel.step()"
   4. Check for changes of output value of model after stepping
      a. e.g. "testingModel.testing_Y.Out1"
      b. the exact variable name is defined within the model class within "[simulink_model_name].h" and should be commented with "/* External outputs */"

[14] http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing.h

c. src/simulink_generated_code/switching_testing_grt_rtw/testing.h[15]

```
/* Class declaration for model testing */
class testingModelClass {
  /* public data and function members */
 public:
  /* External outputs */
  ExtY_testing_T testing_Y;

  /* model initialize function */
  void initialize();

  /* model step function */
  void step();

  /* model terminate function */
  void terminate();

  /* Constructor */
  testingModelClass();

  /* Destructor */
  ~testingModelClass();

  /* Real-Time Model get method */
  RT_MODEL_testing_T * getRTM();

  /* private data and function members */
 private:
  /* Tunable parameters */
  static P_testing_T testing_P;

  /* Real-Time Model */
  RT_MODEL_testing_T testing_M;
};
```

src / motor_control_unit / src / simulink_generated_code / testing_grt_rtw / **testing_main.cpp** Blame

ADDED

```
#include <iostream>

#include <testing.h>

int main(int argc, char **argv)
{
  auto testingModel = testingModelClass();
  testingModel.initialize();
  for(auto i{0u}; i < 10u; ++i)
  {
    testingModel.step();
    std::cout << "Out1 after step(): " << testingModel.testing_Y.Out1 << std::endl;
  }

  testingModel.terminate();

  return 0;
}
```

2.3.1 CMake Project Config and File Collector

For a CMake project generated from Simulink to be ported, necessary C/C++ files provided by MATLAB and Simulink would need to be collected. Manually copying all the files would be tedious and prone to error, therefore, a program to check the generated CMakeLists.txt and automatically globs the files would be a more efficient way to achieve it in a semi-automatic fashion.

A Python file, config_cmake.py[16], has been written in the rt-hil-simulation[17] repository to do just the above.

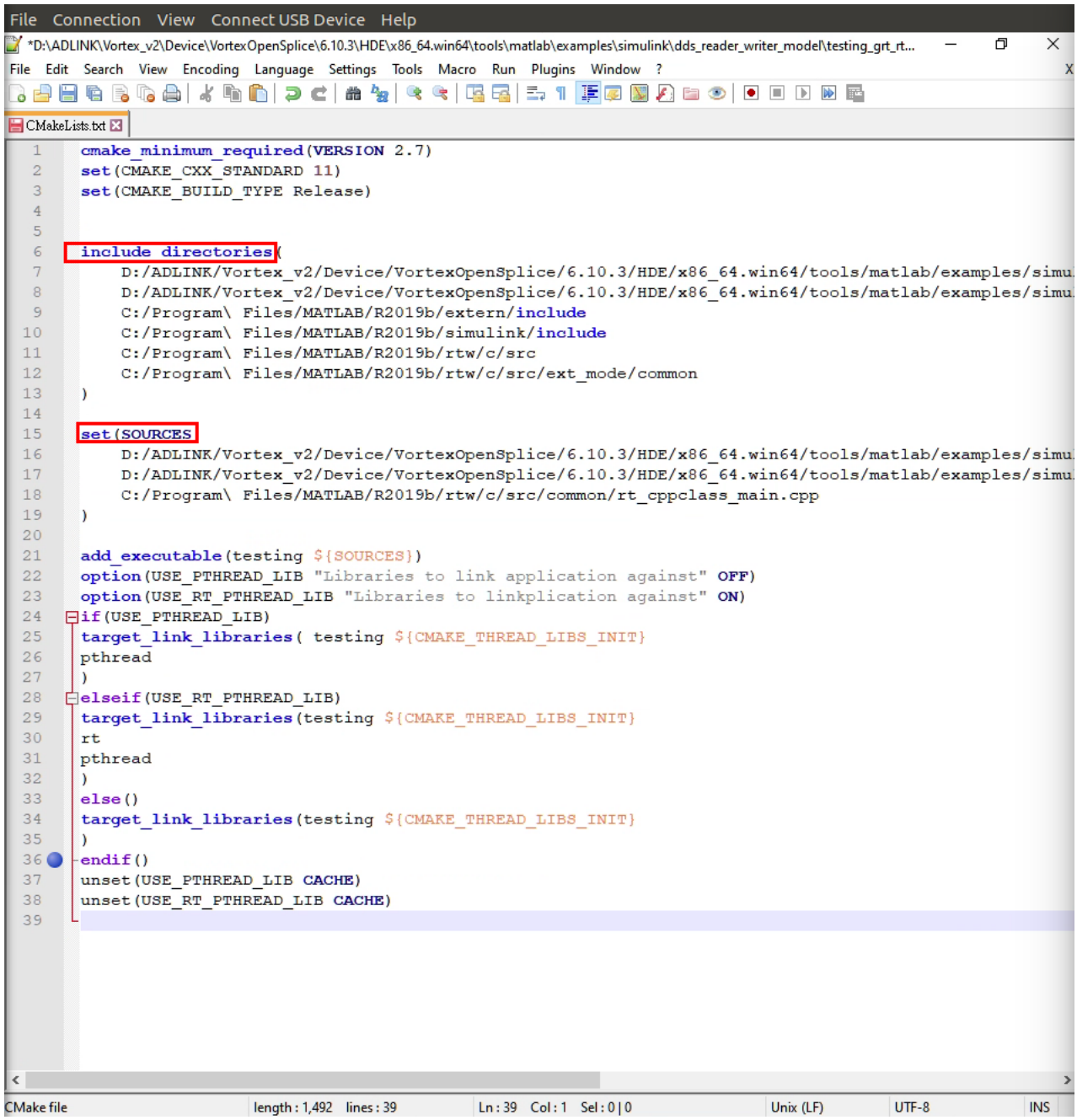
---

15 http://mmslswdev.itri.org.tw/bitbucket/users/A70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing.h

16 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/scripts/config_cmake.py

17 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse

Before getting into the Python file, one can observe what a generated CMakeLists.txt would look like:



By looks of the CMakeLists.txt file, we would be most interested in the directories and files specified from "include_directories" and "SOURCES". Most of these file would need to be found by CMake in order to compile the project. We would assume that the format of the CMakeLists.txt would be the same when the same version of MATLAB Simulink was used.

Next we would create a python file to automate the following tasks:

1. examine the MATLAB generated CMakeLists.txt
2. copy files needed for building the project

3. generate another CMakeLists.txt to compile/build project by itself

2.3.1.1 Examining the MATLAB Generated CMakeLists.txt

To examine the CMakeLists.txt, in the "config_cmake.py" file, the function "FindCmakeParameters()" would store the include directories and source files in a dictionary for use:

```
def FindCmakeParameters(cmake_file_path):
    cmake_parameters = dict()
    with open(cmake_file_path, 'r') as cmake_file:
        for line in cmake_file:
            if "cmake_minimum_required" in line:
                cmake_parameters["cmake_minimum_required"] = \
                    re.split('\(|\)', line)[1]
            if "include_directories" in line:
                include_directories = list()
                line = next(cmake_file)
                while ")" not in line:
                    include_directories.append(removeBadCharactersFromPath(line))
                    line = next(cmake_file)
                cmake_parameters["include_directories"] = include_directories
            if "set(SOURCES" in line:
                sources = list()
                line = next(cmake_file)
                while ")" not in line:
                    sources.append(removeBadCharactersFromPath(line))
                    line = next(cmake_file)
                cmake_parameters["sources"] = sources

    return cmake_parameters


if __name__ == '__main__':
    cmake_parameters = FindCmakeParameters(cmake_directory_path)
```

Here it's basic file reading and detecting certain line. Function "removeBadCharactersFromPath()" just removes spaces, new lines, and \ characters.

We also would need the definitions specified in the "defines.txt" that is generated by MATLAB:

```
def FindCmakeDefinitions(definition_file_path):
    with open(definition_file_path, 'r') as definition_file:
        return [removeBadCharacters(line) for line in definition_file]

if __name__ == '__main__':
    cmake_parameters["definitions"] = FindCmakeDefinitions(sys.argv[1] + "defines.txt")
```

2.3.1.2 Copying Files Needed for Building the Project

Next thing to automate would be to copy the include directories and source files that are stored within the cmake parameters we've found previously. Functions "CopyIncludeDirectories()" and "CopySources()" were written for copying:

```
def CopyIncludeDirectories(directories, target_include_path):
    for directory in directories:
        size = GetSize(directory)
        if size > MAX_DIRECTORY_SIZE:
            print("Warning: copying large directory")
        print("Copying directory with size {} kbytes from {}" \
            .format(size // 1000, directory))
        src_dir = directory
        dst_dir = target_include_path + removeLetterFromPath(directory)
        if os.path.isdir(dst_dir):
            shutil.rmtree(dst_dir)
        shutil.copytree(src_dir, dst_dir)


def CopySources(sources, target_source_path):
    for source in sources:
        size = GetSize(source)
        if size > MAX_DIRECTORY_SIZE:
            print("Warning: copying large file")
        print("Copying file with size {} kbytes from {}" \
            .format(size // 1000, source))
        src_name = source
        dst_name = target_source_path + source.split('/')[-1]
        shutil.copy2(src_name, dst_name)


if __name__ == '__main__':
    cmake_parameters = FindCmakeParameters(cmake_directory_path)

    current_path = os.getcwd()
    target_path = current_path + '/cmake_project/'
    target_include_path = target_path + '/include/'
    target_source_path = target_path + '/src/'

    CopyIncludeDirectories(cmake_parameters["include_directories"], target_include_path)
    CopySources(cmake_parameters["sources"], target_source_path)
```

2.3.1.3 Generating Another CMakeLists.txt to Compile/Build Project by itself

Lastly, a new simple CMakeLists.txt should be provided to build the target library with all the copied directories and files. This would be achieved by function "WriteCmakeFile()":

```
def WriteCmakeFiles(parameters, target_path):
    with open(target_path + 'CMakeLists.txt', 'w') as cmake_file:
        cmake_file.write("cmake_minimum_required({})\n" \
            .format(parameters["cmake_minimum_required"]))
        cmake_file.write("project(simulink_generated_code)\n\n")
        for definition in parameters["definitions"]:
            cmake_file.write("add_definitions(-D{})\n".format(definition))
        cmake_file.write('\n')
        cmake_file.write("add_executable(main\n")
        for source in parameters["sources"]:
            cmake_file.write('  src/' + source.split('/')[-1] + '\n')
        cmake_file.write(")\n\n")
        cmake_file.write("target_include_directories(main\n")
        cmake_file.write("  PUBLIC\n")
        for directory in parameters["include_directories"]:
            cmake_file.write('  include/' + removeLetterFromPath(directory) + '\n')
        cmake_file.write(")\n\n")

if __name__ == '__main__':
    cmake_parameters = FindCmakeParameters(cmake_directory_path)

    current_path = os.getcwd()
    target_path = current_path + '/cmake_project/'
    target_include_path = target_path + '/include/'
    target_source_path = target_path + '/src/'

    CopyIncludeDirectories(cmake_parameters["include_directories"], target_include_path)
    CopySources(cmake_parameters["sources"], target_source_path)

    WriteCmakeFile(cmake_parameters, target_path)
```

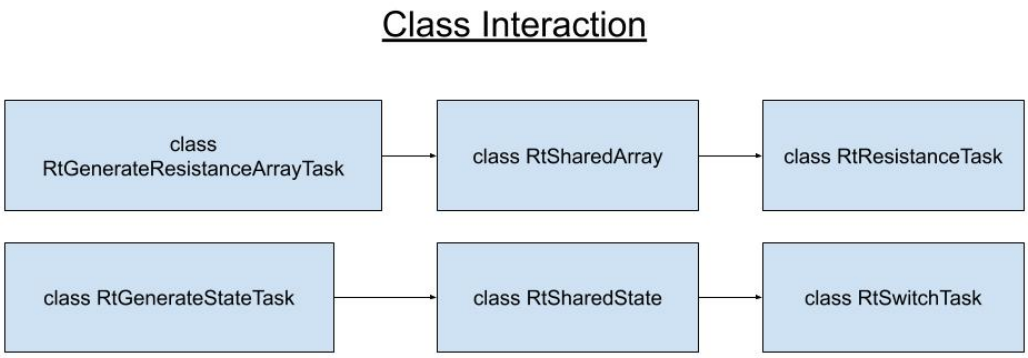
This python script should create a folder to allow for independent compilation. A transmitted cmake project configured by this script can be successfully compiled:

```
d400@u16-01-RTComputer:~/cmake_project$ ls
CMakeLists.txt  include  originalFilePaths.txt  src
d400@u16-01-RTComputer:~/cmake_project$ mkdir build
d400@u16-01-RTComputer:~/cmake_project$ cd build
d400@u16-01-RTComputer:~/cmake_project/build$ cmake .. && make
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/d400/cmake_project/build
Scanning dependencies of target main
[ 25%] Building CXX object CMakeFiles/main.dir/src/testing.cpp.o
[ 50%] Building CXX object CMakeFiles/main.dir/src/testing_data.cpp.o
[ 75%] Building CXX object CMakeFiles/main.dir/src/rt_cppclass_main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
d400@u16-01-RTComputer:~/cmake_project/build$
```

2.4 Resource Sharing Among Xenomai RT Tasks

Scenario:

1. One Xenomai RT Task running Simulink model and generate values to write to Pickering PXI interface card
2. One Xenomai RT Task awaiting value to be written to Pickering PXI interface card
3. A shared resource for both Xenomai RT tasks



We will focus on RtGenerateStateTask, RtSharedState, and RtSwitchTask

Steps

1. Refactor code so that each Xenomai RT Task and regular task can be instantiated as a class
    a. RtGenerateStateTask.h[18], derived from
        i. RtPeriodicTask.h[19]
    b. RtSwitchTask.h[20], derived from
        i. PxiCardTask.h[21] (Non Xenomai RT Task)
        ii. RtPeriodicTask.h[22]
2. Create a shared resource with thread safe mechanisms
    a. RtSharedState.h[23]
        i. includes RT_MUTEX
3. Declared the shared resource with a shared_ptr

---

18 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtGenerateStateTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

19 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtPeriodicTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

20 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtSwitchTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

21 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/PxiCardTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

22 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtPeriodicTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

23 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtSharedState.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

   a. testing_main.cpp[24]: line 16
4. Add shared_ptr of the shared resource to the Xenomai RT Task running Simulink model
   a. RtGenerateStateTask.h[25]: line 16
   b. testing_main.cpp[26]: line 43
5. Add shared_ptr of the shared resource to the Xenomai RT Task accessing the Pickering PXI interface card
   a. RtSwitchTask.h[27]: line 15
   b. testing_main.cpp[28]: line 57

Reference:

- src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp[29]

2.5 Xenomai RT Running Peak CAN

Scenario: CAN with one PEAK CAN USB node and one PEAK CAN PCI FD (port #2) node with a termination block in between

1. Ensure Successful CAN communication (from USB to PCI FD, and vice versa)
   a. Add compilable skeleton code for transmit and receive into CMake Project
      i. commits 63b4d9f3418[30] and a469ef9531b[31]
   b. main CAN functions
      i. LINUX_CAN_Open()
      ii. CAN_Init()
      iii. CAN_Close()
      iv. CAN_Read()
      v. CAN_Write()
      vi. see peak-linux-driver-8.9.0/lib/libpcan.h[32] for function headers and peak-linux-driver-8.9.0/lib/src/libpcan.c[33] for function definitions
   c. Code for one node to receive and the other to transmit
      i. commit 8593ca2b0b1[34]
   d. refactor to class PeakCanTask

---

24 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

25 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtGenerateStateTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

26 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

27 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/RtSwitchTask.h?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

28 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

29 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=f081ce54bcb4164c3bbde24e3b7de020155158ad

30 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits/a469ef9531b42234986dc3ee252f64e441c8162f#src/motor_control_unit/CMakeLists.txt

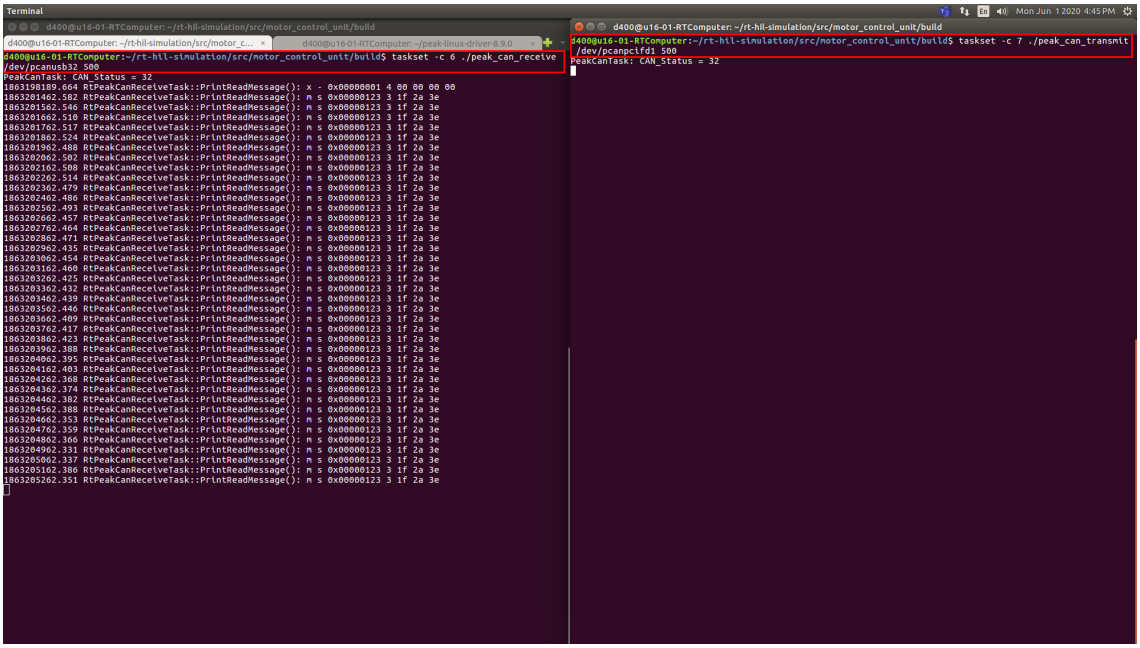
31 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits/8593ca2b0b1cffd84ad1df19bb268ca27a7d337d

32 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/peak-linux-driver-8.9.0/browse/lib/libpcan.h

33 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/peak-linux-driver-8.9.0/browse/lib/src/libpcan.c

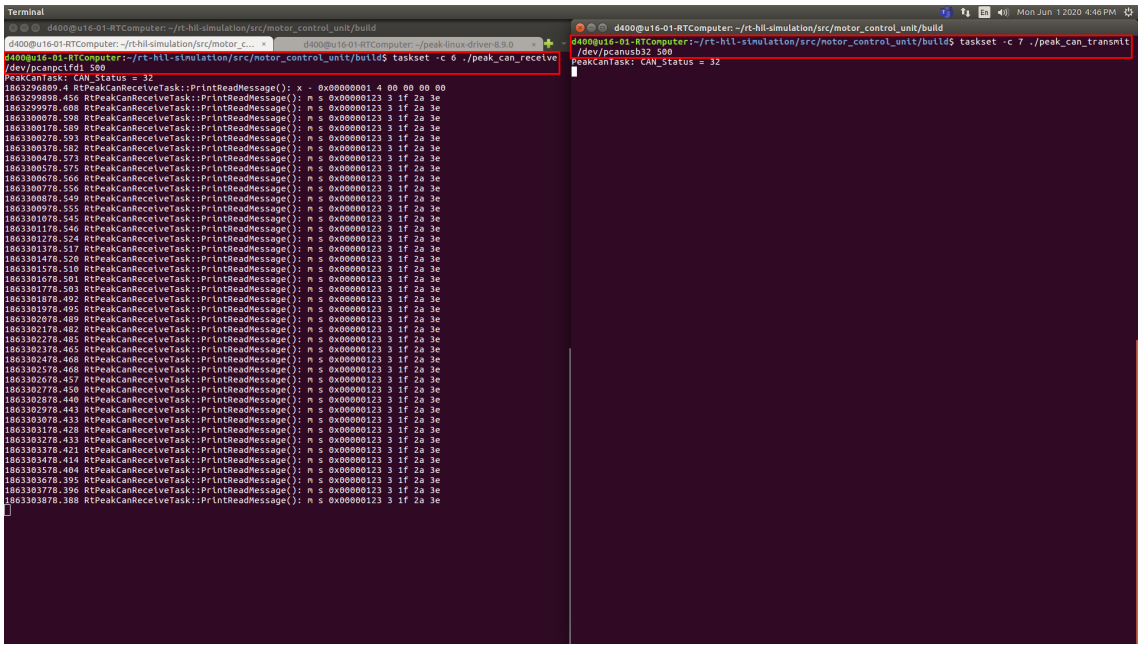
34 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits/9b1e68f2eecc2c8a1b99ae00ea76169e9b07f98d#src/motor_control_unit/src/peak_can_code/peak_can_receive.cpp

   i. see files PeakCantask.cpp[35] and PeakCanTask.h[36]
2. Inherit from RtPeriodicTask to be a Xenomai RT Task
   a. Making the transmit node a Xenomai RT task
      i. writing a new class RtPeakCanTransmitTask, see commit 8593acea3f8[37]
      ii. writing a new class RtPeakCanReceiveTask, see commit 797956429f4[38]
3. Successful Transmission
   a. from PEAK CAN PCI FD to PEAK CAN USB



35 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/peak_can_code/PeakCanTask.cpp
36 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/peak_can_code/PeakCanTask.h
37 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits/5d1ad938d74b266818a10a8072426c39ca9ab3dc#src/motor_control_unit/src/peak_can_code/RtPeakCanTransmitTask.cpp
38 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits/86c4c41cbf59e9eaa1966cfe0053280251107b53

b. Transmission from PEAK CAN USB to PEAK CAN PCI FD

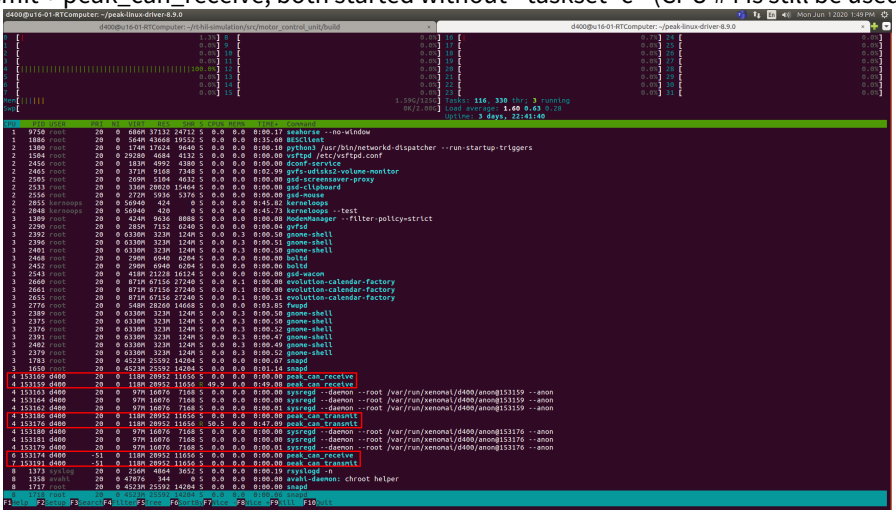


2.6 Xenomai RT Task dedicate CPU core (CPU Affinity)

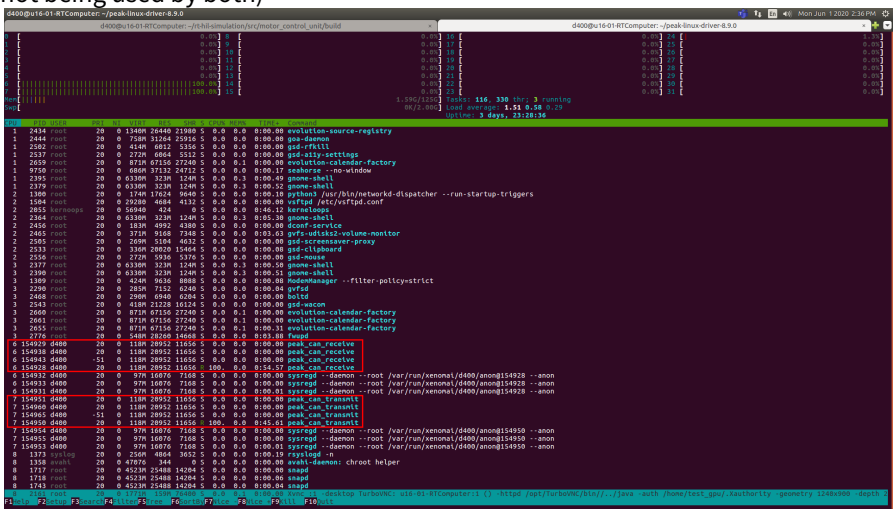
Procedure

1. Call rt_task_set_affinity() for your Xenomai RT Task
2. check /etc/default/grub
    a. for a Xenomai kernel to run Xenomai Tasks running on particular cores, a kernel boot option will be specified in this file for it
    b. "xenomai.supported_cpus=0x000000F0"
        i. 0x000000F0 is in hexadecimal format, its binary representation is (0000 0000 0000 0000 0000 0000 1111 0000)$_2$
        ii. Whichever bit is set (if it's 1), the CPU core will be dedicated for Xenomai. For example, if the number is
            1. (0000 0000 0000 0000 0000 0000 0000 0001)$_2$, then CPU core #0 will be dedicated for Xenomai
            2. (1111 1111 1111 1111 1111 1111 1111 1110)$_2$, then CPU core #1~31 will be dedicated for Xenomai
        iii. This shows that the computer will activate CPU core #4~7 for Xenomai
3. Run with `taskset -c`
    a. if `taskset -c` is not used, any Xenomai task would use the lowest "xenomai.supported_cpus" as main process, and having its children tasks to be spawned inside the CPU core set above.
    b. So for a xenomai program/task to run solely on one CPU, one would execute the program with `taskset -c` and also have the above cpu affinity

c. peak_can_transmit + peak_can_receive, both started without `taskset -c` (CPU #4 is still be used by both processes)



d. peak_can_transmit + peak_can_receive, both started with `taskset -c` on different cores (CPU #4 is not being used by both)



4. observe with htop

1. the lowest number CPU core out of xenomai.supported_cpus will always be used when spawning a Xenomai RT Task
    a. if "xenomai.supported_cpus=0x000000F0", then CPU core #4 will always be used when any Xenomai RT Task is spawned.
2. sometimes the RT task requires very little CPU usage that it wouldn't show it on the top (with all the CPU cores and usage). For a better monitor result, observe the cpu usage with percentage in the list below by searching the actual process/thread name
3. for convenience, you can change the CPU numbering to start with 0 by the following steps
    a. Press F2 to enter "setup"
    b. Use Arrow keys to select "Display options"
    c. Use Arrow keys to navigate to "Count CPUs from 0 instead of 1"
    d. Press space to activate it (it will show a "x" when it's activated)
    e. Press F10 to be done

- observe with /proc/xenomai/sched/rt/threads
    a. when Xenomai RT_TASK are spawned, this file will show threads with its CPU affinity

2.7 Xenomai RT Task Running Pickering PXI Card

1. make all PXI variables (see non_precision_resistor_controller_main.cpp[39], lines 20-32)
2. declare Xenomai variables
    a. RT_TASK
    b. RTIME
3. define a function/routine for Xenomai RT to execute (lines 42-77)
    a. be sure to open the card before the infinite loop within this function/routine
4. spawn a Xenomai RT task to execute the function (lines 145-148)

2.8 Xenomai RT Task Running Simulink Model

For the following steps, a working example (testing_main.cpp) can be found from the rt-hil-simulation Bitbucket reposiroty: commit 442ffd43f86, src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp[40]

To have Simulink Model running with Xenomai RT Task one can follow the steps below:

1. declare Xenomai RT_TASK
    a. "#include <alchemy/mutex.h>"
        i. xenomai api reference Xenomai 3.1 Alchemy API[41]
    b. declare a RT_TASK as global variable (testing_main.cpp: line 41)
        i. "RT_TASK rtSwitchStateTask"
2. declare Simulink Model as global variable
    a. "static auto testingModel = testingModelClass()" (testing_main.cpp: line 47)
3. write a function that would have the Xenomai task execute
    a. "void GenerateSwitchStateRoutine(void*)" (testing_main:cpp: line 51-65)
        i. the function has to return void
        ii. the function needs to have void* as function parameter
        iii. include an infinite loop if the Xenomai task will be a periodic task, in this case it is a periodic task.
            1. make sure the Simulink model is initialized before the infinite loop
            2. "testingModel.step()"
            3. "rt_task_wait_period(NULL)"
                a. this will cause the Xenomai periodic task to wait for a time period and then enter the while loop again.

9 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/non_precision_resistor_controller_main.cpp?at=5e0958eaf28b8b535c8b658d51757f16b97c1e64

0 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=442ffd43f86e6ebcc5e78329eeb75b4e540d835a

41 https://xenomai.org/documentation/xenomai-3/html/xeno3prm/group__alchemy.html

b.
```
void GenerateSwitchStateRoutine(void*)
{
  testingModel.initialize();

  while(true)
  {
    testingModel.step();

    rtSharedState.Set(!rtSharedState.Get());

    rt_task_wait_period(NULL);

    printf("state is: %s\n", rtSharedState.Get() ? "true" : "false");
  }
}
```

4. write code that would create a Xenomai task to execute the function in step 3
    a. "int StartGenerateSwitchStateRoutine()"
        i. "rt_task_create(&rtSwitchStateTask, "GenerateSwitchStateRoutine", kTaskStackSize, kMediumTaskPriority, kTaskMode)"
            1. creates a task that uses global RT_TASK object "rtSwitchStateTask"
            2. give a name "GenerateSwitchStateRoutine"
            3. "kTaskStackSize", "kMediumTaskPriority", and "kTaskMode" are configurations for the task, see Xenomai 3.1 API - rt_task_create()[42]
        ii. "rt_task_set_period(&rtSwitchStateTask, TM_NOW, rt_timer_ns2ticks(1000000))"
            1. makes a task periodic by setting its period
            2. starts "TM_NOW"
            3. period is "rt_timer_ns2ticks(1000000)"
                a. 1000000 nanoseconds
                b. rt_timer_ns2ticks converts this to a Xenomai RTIME object required for function rt_task_set_period
        iii. rt_task_start(&rtSwitchStateTask, &GenerateSwitchStateRoutine, NULL)"
            1. start the task to execute the function written from step 3, "&GenerateSwitchStateRoutine", with a "&" appended in the front
            2. "NULL" because the function we defined takes in void* arguments.

b.
```
int StartGenerateSwitchStateRoutine()
{
  int e1 = rt_task_create(&rtSwitchStateTask, "GenerateSwitchStateRoutine",
    kTaskStackSize, kMediumTaskPriority, kTaskMode);
  int e2 = rt_task_set_periodic(&rtSwitchStateTask, TM_NOW, rt_timer_ns2ticks(1000000));
  int e3 = rt_task_start(&rtSwitchStateTask, &GenerateSwitchStateRoutine, NULL);

  if(e1 | e2 | e3)
  {
    printf("Error launching StartGenerateSwitchStateRoutine(). Exiting.\n");
  }
}
```

5. include an infinite loop in int main
    a. without the infinite loop, the program will exit automatically once executed. This is not what we want. We want the program to run until user terminate it (with ctrl + c or kill -p)

42 https://xenomai.org/documentation/xenomai-3/html/xeno3prm/group__alchemy__task.html#ga03387550693c21d0223f739570ccd992

   b. Reason: int main will be the parent thread, while the Xenomai RT task will be its child task. If int main exit prematurely, it will kill all child processes, including the Xenomai RT task, even if the RT task have an infinite loop.
6. (Optional) Include a termination handler
   a. testing_main.cpp[43]: line 80-88
   b. testing_main.cpp[44]: line 92-97

Reference: src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp[45]

2.9 Xenomai Tutorial - Creating a Simple Program for Xenomai

2.9.1 **Finding Xenomai Package with CMake**

By default, CMake doesn't have an automatic way to find Xenomai. But there's people who wrote cmake files that would have CMake to search through the default installation path of Xenomai and load the package. A file like this has been added in repository rt-hil-simulation(see page 40). Configurations are written in CMakeLists.txt with `cmake` instructions to generate Makefile's, and then later to run `make` for compilation. If you want to compile a separate CMake project the steps would be as follows:

1. Download "FindXenomai.cmake" (cmake/FindXenomai.cmake[46])
2. Create a "cmake" directory in the same folder where the CMakeLists.txt is located
   a. see https://cmake.org/cmake/help/latest/guide/tutorial/index.html for CMake Tutorial
3. Put "FindXenomai.cmake" within the "cmake" directory
4. set "CMAKE_MODULE_PATH" to where the "cmake" directory is located
   a. Ensure the following line is in your CMakeLists.txt
      i. "set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${CMAKE_CURRENT_SOURCE_DIR}/cmake)"
         1. see CMakeLists.txt[47]
         2. CMAKE_MODULE_PATH and CMAKE_CURRENT_SOURCE_DIR are predefined in any cmake projects
      ii. This line will be a little bit different if your CMakeLists.txt and cmake are not in the same directory
         1. see src/xenomai_code/CMakeLists.txt[48]
5. find Xenomai package
   a. ensure the following line is in your CMakeLists.tx
      i. "find_package(Xenomai 3 REQUIRED)"

---

43 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=442ffd43f86e6ebcc5e78329eeb75b4e540d835a
44 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=442ffd43f86e6ebcc5e78329eeb75b4e540d835a
45 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/simulink_generated_code/switching_testing_grt_rtw/testing_main.cpp?at=442ffd43f86e6ebcc5e78329eeb75b4e540d835a
46 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake/FindXenomai.cmake
47 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/CMakeLists.txt
48 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/xenomai_code/CMakeLists.txt

   b. if you set the correct CMAKE_MODULE_PATH, CMake will automatically go into your "cmake" directory, find a file that's named "FindXenomai.cmake" and load Xenomai package for your CMake project to use
      i. "CMake searches for a file called "Find<package>.cmake" in the "CMAKE_MODULE_PATH" followed by the CMake installation. If the file is found, it is read and processed by CMake." (see cmake find_package()[49])
   c. Note: "FindXenomai.cmake" file also sets some variables such as "XENOMAI_FOUND", "XENOMAI_INCLUDE_DIRS", and "XENOMAI_LIBRARIES", etc. These are to be used for compiling a Xenomai program within a CMake project. See line 171[50] and line 66[51] of FindXenomai.cmake[52].
   d. (Debug) If you encountered an error that says "Could NOT find Xenomai (missing: XENOMAI_XENO_CONFIG)" such that

```
d400@Training02:~/xenomai-practice/build$ cmake ..
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMake Error at /usr/share/cmake-3.10/Modules/FindPackageHandleStandardArgs.cmake:137 (message):
  Could NOT find Xenomai (missing: XENOMAI_XENO_CONFIG) (Required is at least
  version "3")
Call Stack (most recent call first):
  /usr/share/cmake-3.10/Modules/FindPackageHandleStandardArgs.cmake:378 (_FPHSA_FAILURE_MESSAGE)
  cmake/FindXenomai.cmake:186 (find_package_handle_standard_args)
  CMakeLists.txt:6 (find_package)


-- Configuring incomplete, errors occurred!
See also "/home/d400/xenomai-practice/build/CMakeFiles/CMakeOutput.log".
```

      i.
      ii. then make sure to take care of Xenomai's environment variables Installing and Running Xenomai on the Custom-Built Linux Kernel(see page 65), step 2c.

6. add a target for CMake to compile
   a. add_executable([executable name] [source files])
      i. add_executable(xenomai_task src/xenomai_task.cpp)
         1. the source files' paths are relative to the CMakeLists.txt file
7. link the Xenomai Libraries
   a. "target_link_libraries([target_name] ${XENOMAI_LIBRARIES})"
      i. target_link_libraries(xenomai_task ${XENOMAI_LIBRARIES})
8. include the Xenomai directories for header files
   a. "target_include_directories([target_name] SYSTEM PUBLIC ${XENOMAI_INCLUDE_DIRS})"
9. Create a C++/C file that have "#include <alchemy/task.h>" and check for successful compilation.

---

49 https://cmake.org/cmake/help/v3.0/command/find_package.html
50 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake/FindXenomai.cmake#171
51 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake/FindXenomai.cmake#66
52 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake/FindXenomai.cmake#66

```
#include <alchemy/task.h>

int main(int argc, char *argv[])
{
    return 0;
}
```

a. e.g.: `vim src/xenomai_task.cpp`

2.9.2 First Simple Xenomai RT Periodic Task

Xenomai has different "flavors" to choose from, mainly for others who have programmed their RT programs from other infrastructure to migrate over. Here we will use the native Xenomai flavor, named Alchemy flavor. In the native, Alchemy flavor, Xenomai can spawn RT tasks and have communications among the tasks (very similar to multi-threading) to execute programmed function. For our first Xenomai task, we would create a periodic task that will periodically executes its programmed function within a specified time period. Here's the general procedure for writing it:

1. Define the routine for your task

```
void Routine(void*)
{
    for (;;)
    {
        printf("hi\n");
        rt_task_wait_period(NULL);
    }
}
```

   a. must be a function that returns void
   b. must be a function that has void* as input
   c. must have an infinite for loop and a function call "rt_task_wait_period()" if it's a periodic task
2. Start the Routine

```
RT_TASK rtTask;

auto constexpr kStackSize = 0;
auto constexpr kPriority = 99;
auto constexpr kMode = 0;
auto constexpr kOneSecond = 1000000000.0;

int main(int argc, char *argv[])
{
    rt_task_create(&rtTask, "RtTask", kStackSize, kPriority, kMode);
    rt_task_set_periodic(&rtTask, TM_NOW, rt_timer_ns2ticks(kOneSecond));
    rt_task_start(&rtTask, Routine, NULL);

    for (;;)
    {}

    return 0;
}
```

   a. Ensure parent thread (int main()) and its children RT Tasks (the routines) have infinite loop
   a. if "int main" doesn't have an infinite loop, it will exit and kill all children RT Tasks
   b. whenever rt_task_start() is called, the program will become a parent to the child RT Task
3. Enable program to handle "ctrl + c" termination signal (Optional but recommended)

```
int terminationHandler(int signal)
{
    printf("Caught ctrl + c termination signal. Exiting.\n");
    exit(1);
}

int main(int argc, char *argv[])
{
    struct sigaction action;
    action.sa_handler = terminationHandler;
    sigemptyset(&action.sa_mask);
    action.sa_flags = 0;
    sigaction(SIGINT, &action, NULL);

    return 0;
}
```

4. see src/xenomai_code[53]
5. Observe Xenomai RT Task
   a. htop
   b. `cat /proc/xenomai/sched/rt/threads`

References:

- Xenomai 3 Online API Documentation https://xenomai.org/documentation/xenomai-3/html/xeno3prm/group__alchemy.html

53 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/xenomai_code

- Simple Xenomai RT Task Code: src/xenomai_code/test_rt.cpp[54]
- CMake File for Compiling Xenomai RT:
  - src/xenomai_code/CMakeLists.txt[55]
  - cmake/FindXenomai.cmake[56]
- Xenomai 2.4 Exercises (older version, but can be used as a reference): xenomai exercises[57]
- xenomai-3 tutorial from online resources
  - 

54 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/xenomai_code/test_rt.cpp
55 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/xenomai_code/CMakeLists.txt
56 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/cmake/FindXenomai.cmake
57 http://www.cs.ru.nl/lab/xenomai/exercises_xenomai2.4/

3 Accessing Pickering PXI Interface Card with C++

1. install repository pickering-linux-driver[58]
    a. forgot how to install, will add details later
    b. I think it's `sudo ./install.sh`
    c. and then `pilpxi_setup.sh`
2. include pilpxi header
    a. "#include <Pilpxi.h>"
3. find pilpxi package in CMakeLists.txt
4. Functions to access a PXI non-precision resistance interface card (see src/pickering_code/non_precision_resistor_controller_main.cpp @ commit f6551c76d5b[59])
    a. "PIL_CountFreeCrads(&numOfFreeCards)"
    b. "PIL_FindFreeCards(numOfFreeCards, buses, devices)"
    c. "PIL_OpenSpecifiedCard(buses[i], devices[i], &cardNum)"
    d. "PIL_CardId(cardNum, id)"
    e. "PIL_ReadSub(cardNum, 1, data)"
    f. "PIL_WriteSub(cardNum, 1, data)"
    g. "PIL_CloseSpecifiedCard(cardNum)"

All PIL related functions see: Pilpxi.h[60]

58 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/pickering-linux-driver/browse
9 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/pickering_code/non_precision_resistor_controller_main.cpp?at=f6551c76d5b2557d0d82de1a3cf683fa232a7290
60 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/pickering-linux-driver/browse/Pilpxi.h

4 National Instrument NI PCIe-6361 I/O Card

4.1 Documentations

- Data Sheet
  - 

- X Series User Manual
  - 

- X Series Reference Manual

- 

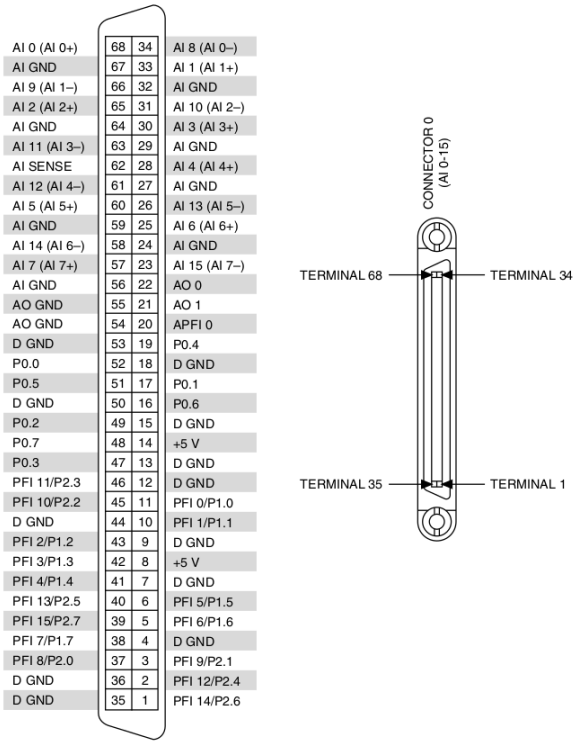
4.2 PWM (Pulse-Width Modulation) Input/Output

Prerequisite: finished Installing National Instrument NI PCIe-6361 Driver(see page 86)

National Instrument PCIe-6361 is capable of outputting and inputting PWM (Pulse-Width Modulation) with its onboard counters and triggers along with a connected breakout board for accessing particular digital lines. It can output or receive PWM signals on the PFI (Programmable Function Interface) lines. These PFI lines are Digital Input Output (DIO) lines and their corresponding terminals are as follows:

- PFI 0/P1.0 → terminal 11
- PFI 1/P1.1 → terminal 10
- PFI 2/P1.2 → terminal 43
- PFI3/P1.3 → terminal 42
- PFI4/P1.4 → terminal 41
- PFI 5/P1.5 → terminal 6
- PFI 6/P1.6 → terminal 5
- PFI 7/P1.7 → terminal 38
- PFI 8/P2.0 → terminal 37
- PFI 9/P2.1 → terminal 3
- PFI 10/P2.2 → terminal 45
- PFI 11/P2,3 → terminal 46
- PFI 12/P2.4 → terminal 2
- PFI 13/P2.5 → terminal 40
- PFI 14/P2.6 → terminal 1
- PFI 15/P2.7 → terminal 39

Figure 9. NI PCIe/PXIe-6361 Pinout

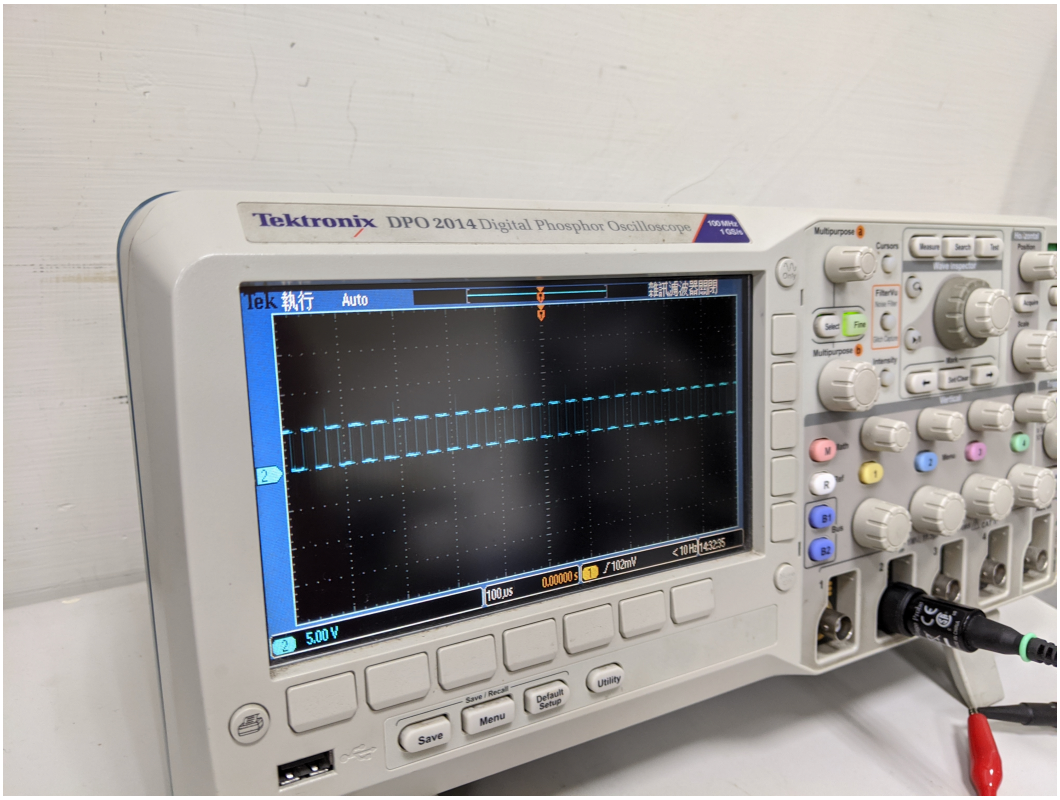


PWM Output

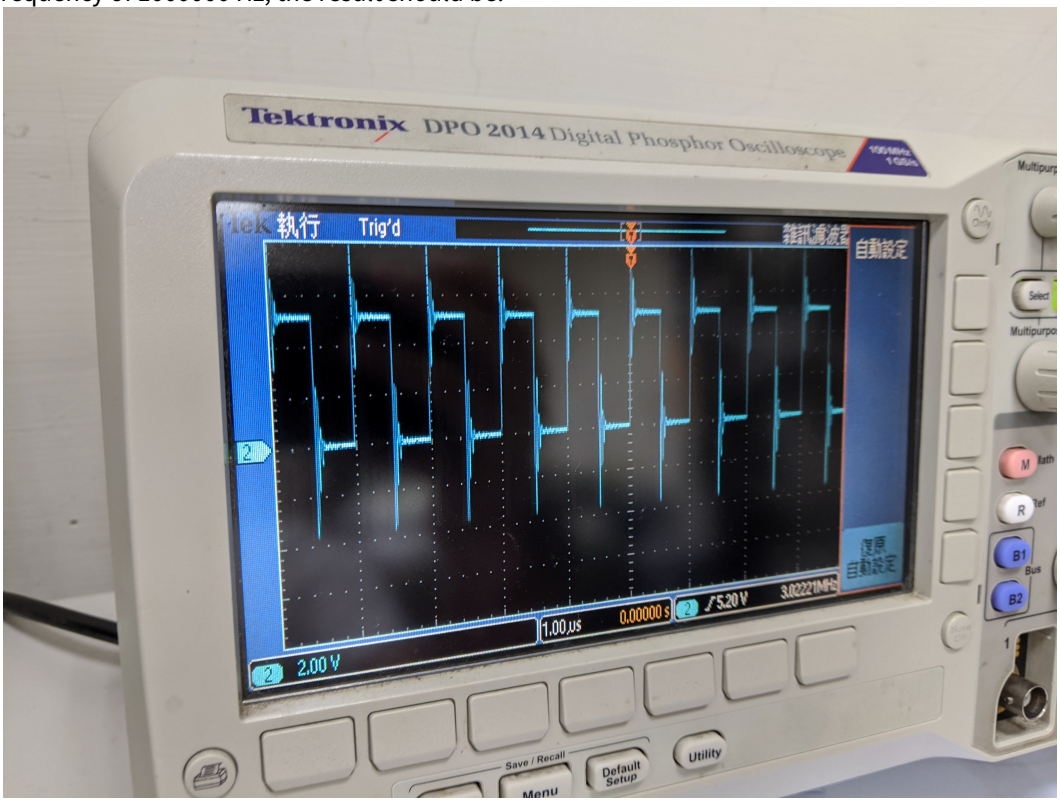
For the NI PCIe-6361 to output PWM signals, we would use files pwm_output_main.cpp[61] and pwm_output.cpp[62], which were ported from the "gpctex7.cpp" examples in the nixseries repository with minor modification.

1. The PWM signals is hard-coded to be outputted to digital line PFI1, which is on terminal 10, and to use Counter #1.
2. Insert a pin or short wire to receive extend the terminal. Make sure to choose a D GND pinout (e.g. terminal 53, 13, etc.) for grounding as well when measuring.
3. Connect an oscilloscope, with its positive on the PFI1 pinout on terminal 10, and ground on the pinout with D GND.
4. Get the bus number and device number for the connected NI PCIe-6361 card by running lsdaq
    a. see "Installing Linux Kernel Module Driver for Linux to Use the PCIe-6361" section in Installing National Instrument NI PCIe-6361 Driver(see page 86)
5. Compile target "pwm_output" from the CMakeLists.txt
6. Run "pwm_output"
    a. e.g. `pwm_output 27 0 45 10000 10`
    b. with a frequency of 10000 Hz, the result on the oscilloscope should be:

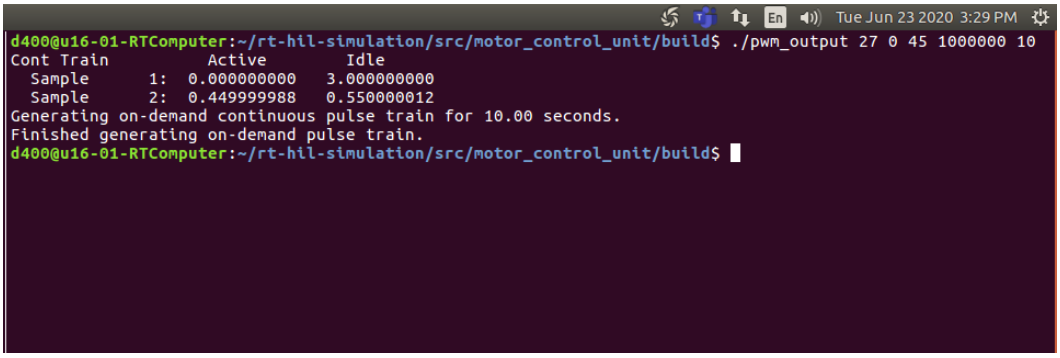
61 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/ni_code/pwm_output_main.cpp
62 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/ni_code/pwm_output.cpp

i. 

c. with a frequency of 1000000 Hz, the result should be:

i. 

ii.

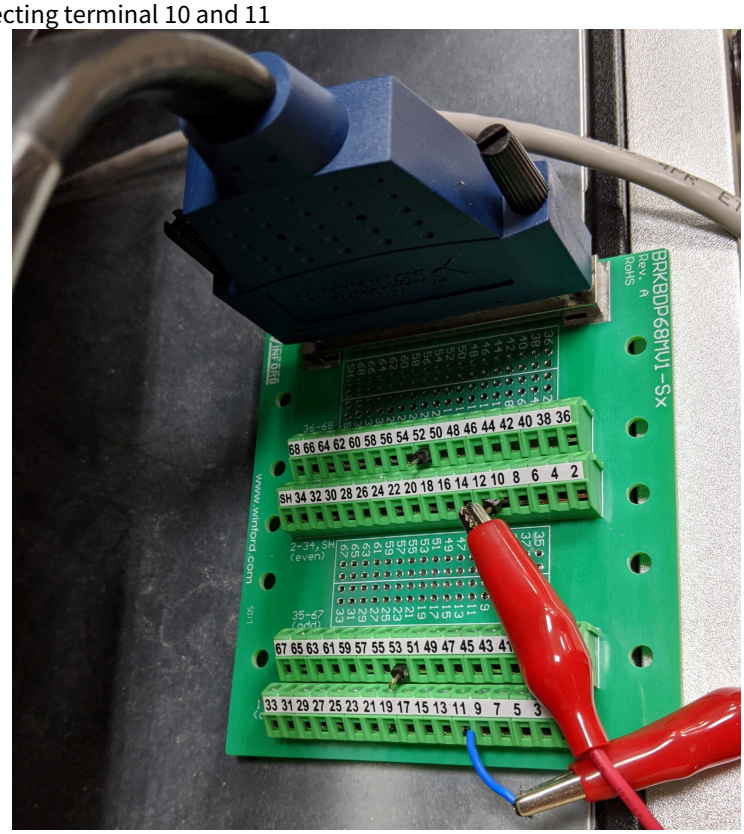


PWM Input

To use NI PCIe-6361 to measure PWM signals, we can use files pwm_input_main.cpp[63] and pwm_input.cpp[64] which are also ported from the example codes from the nixseries repository with minor modification.

1. The terminal to be used for measuring PWM signals have been hard-coded to be line PFI0, terminal 11 and uses Counter #0
2. The PWM output line PFI1 can be connected to PFI0 to measure our generated PWM output signals
   a. connecting terminal 10 and 11

i.



3. Compile "pwm_input" from CMakeLists.txt

63 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/ni_code/pwm_input_main.cpp
64 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/ni_code/pwm_input.cpp

4. Run both "pwm_input" and "pwm_output"
    a. the output from "pwm_input" should be the following, outputting received time of each sample (either a rising edge of a falling edge). Also calculating the duty cycle and frequency based on the timing of the samples.

Make these Xenomai Tasks?

There's probably no need to encapsulate the NI PCIe-6361 codes into Xenomai tasks because the functionality are already running from the card itself; it's already real-time when the card is using an internal sampling clock with dedicated registers handling each programmed tasks without undesired interruptions. And besides, it would be much work with little efficiency with the PWM able to output and measure in a frequency of 100 MHz and a 10 nanoseconds margin of error.

5 OpenSplice DDS

5.1 OpenSplice DDS Evaluation Installation on Ubuntu 18.04

Procedure:

1. Visit site https://www.adlinktech.com/Products/IoT_solutions/Vortex_DDS/Vortex_OpenSplice?lang=en
2. Click on "Software Evaluation"
3. Login by creating a user
4. click "ALL"
5. choose "Vortex OpenSplice Evaluation for Linux kernel 3.0 and above (64-bit), gcc 5.4, x86 chipset
6. fire up a terminal
7. `cd` to the directory where it was downloaded
8. `chmod +x P768-VortexOpenSplice*`
9. `./P768-VortexOpenSplice*`

5.2 OpenSplice DDS Node.JS API

OpenSplice DDS Evaluation Version contains a node.js API for extended communication in this applications written in this language.

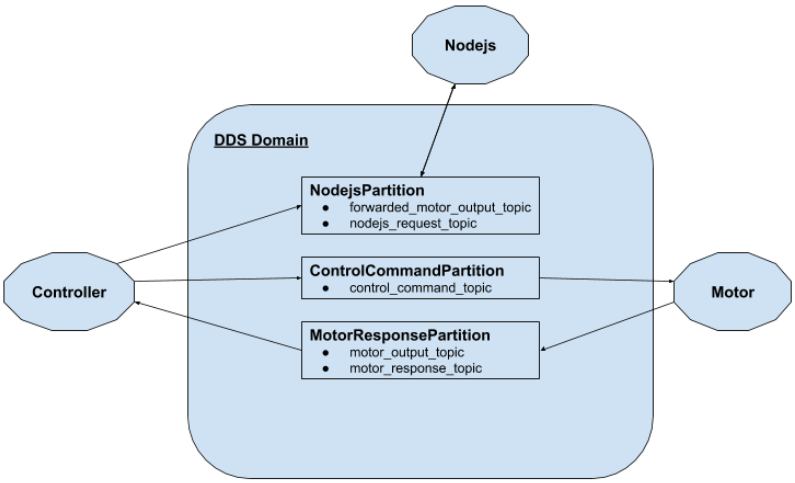
A TypeScript DDS bridge ("ddsBridge.ts") has been implemented for bridging connection between nodejs applications and other DDS applications.

Procedures to using ddsBridge

1. Install OpenSplice DDS Evaluation from their site (see OpenSplice DDS Evaluation Installation on Ubuntu 18.04(see page 52))
    a. note: the installation file can only be installed on the machine that downloads it.
2. clone repository dds-motor-controller[65]
    a. `git clone http://mmslswdev.itri.org.tw/bitbucket/scm/~a70572/dds-motor-controller.git`
3. enter directory "dds-motor-controller/src/motor_control_unit/src/nodejs_code/dds_bridge"
4. `source ${OSPL_HOME}/release.com`
    a. "${OSPL_HOME}" is where the installation of OpenSplice DDS
    b. example path of ${OSPL_HOME}= /ADLINK/Vortex_v2/Device/VortexOpenSplice/6.10.3/HDE/x86_64.linux
5. `npm install ${OSPL_HOME}/tools/nodejs/vortexdds-6.10.0.tgz`
6. in file "ddsBridge.ts", use "class DDSBridge" and function "sendNodejsRequestToController()" as example/template for using the API
7. install Node.JS
    a. TODO
8. install TypeScript
    a. TODO
9. `npm run tsc`
10. `node ddsBridge.js`

65 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/dds-motor-controller/browse

Overview of Applications using DDS



- Each application ("Nodejs", "Controller", and "Motor") has partition that it subscribes and publishes to. Within each partition are the topics that will be shared among applications subscribing/publishing to the corresponding partition(s).
- Nodejs request will first be sent to controller, then controller will request from motor the output. Then upon receiving the output from motor, controller will forward the output to nodejs.

Work Logs

- 2020-03-12 Work Log - DDS's Node.js API[66]
- 2020-03-13 Work Log - Bridging C++ & Node.js with DDS[67]
- 2020-03-17 Work Log - Updating Node.js and Learning TypeScript[68]
- 2020-03-18 Rewrite DDS's Nodejs API with TypeScript[69]
- 2020-03-25 Work Log - DDS Nodejs API[70]

5.3 OpenSplice DDS Simulink Guide

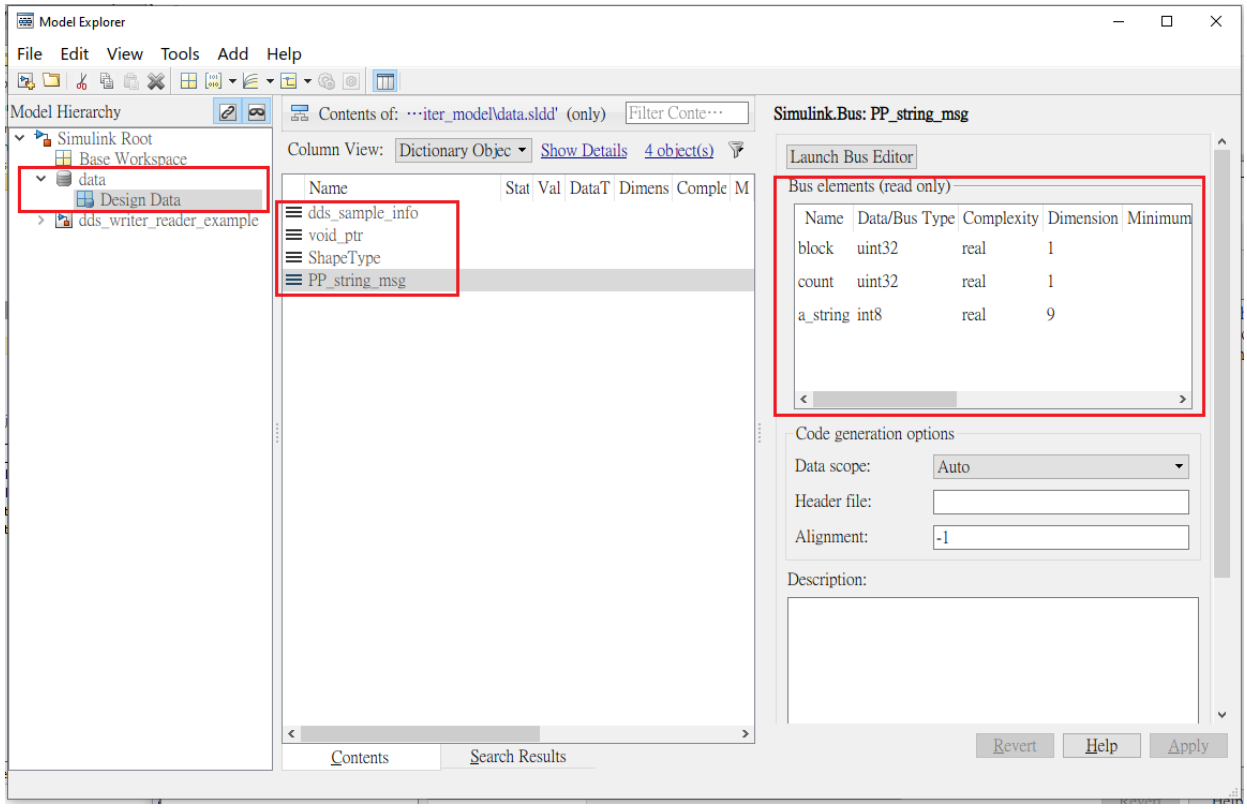
Simulink Bus to DDS Topic Mapping

Simulink data is represented in buses whose types are not compatible with DDS topic data types. When using the Simulink Vortex DDS library, the user must create Simulink buses that will be mapped to DDS topic types. For this, the Vortex DDS Reader, Writer, and Topic blocks have block parameters that require a Simulink bus type.

On data writes, the Simulink bus types are converted to DDS topic types and on data reads, the DDS topic types are converted to Simulink bus types.

The user can generate/create the Simulink bus definitions by either generating them from an IDL file, or by using the Simulink bus editor.

66 https://drive.google.com/open?id=1iUP1IL3-dJX8yEDTpwq62BPkgnTzR4kCHovF_qHFT8Y
67 https://drive.google.com/open?id=1cEeotTBTc0X4ZbqPyYuNEmnj5civ0i9lUqRaPuSDvig
68 https://drive.google.com/open?id=1THJpywMeS7fcZmYWmfDjfC00tx55QA_M4SD_HdqwLuA
69 https://docs.google.com/document/d/1GpUV7-McKQCMj647LMyGxhsVeAh5Z6_KopM7y4ELvnM/edit?usp=sharing
70 https://drive.google.com/open?id=1BzgsWfLeUkYfQQMwaGiJfH07WEZph8Xb3Loz2cK1FtQ



after loading C:\Program Files\ADLINK\Vortex_v2\Device\VortexOpenSplice\6.10.3\HDE\x86_64.win64\tools\matlab\examples\simulink\dds_reader_writer_model\data.sldd

If no sldd file (Simulink Data Dictionary file) can be loaded, then an IDL file can be used to import a Simulink bus with function idlImportSl(IDLFILENAME, DICTIONARYFILE)

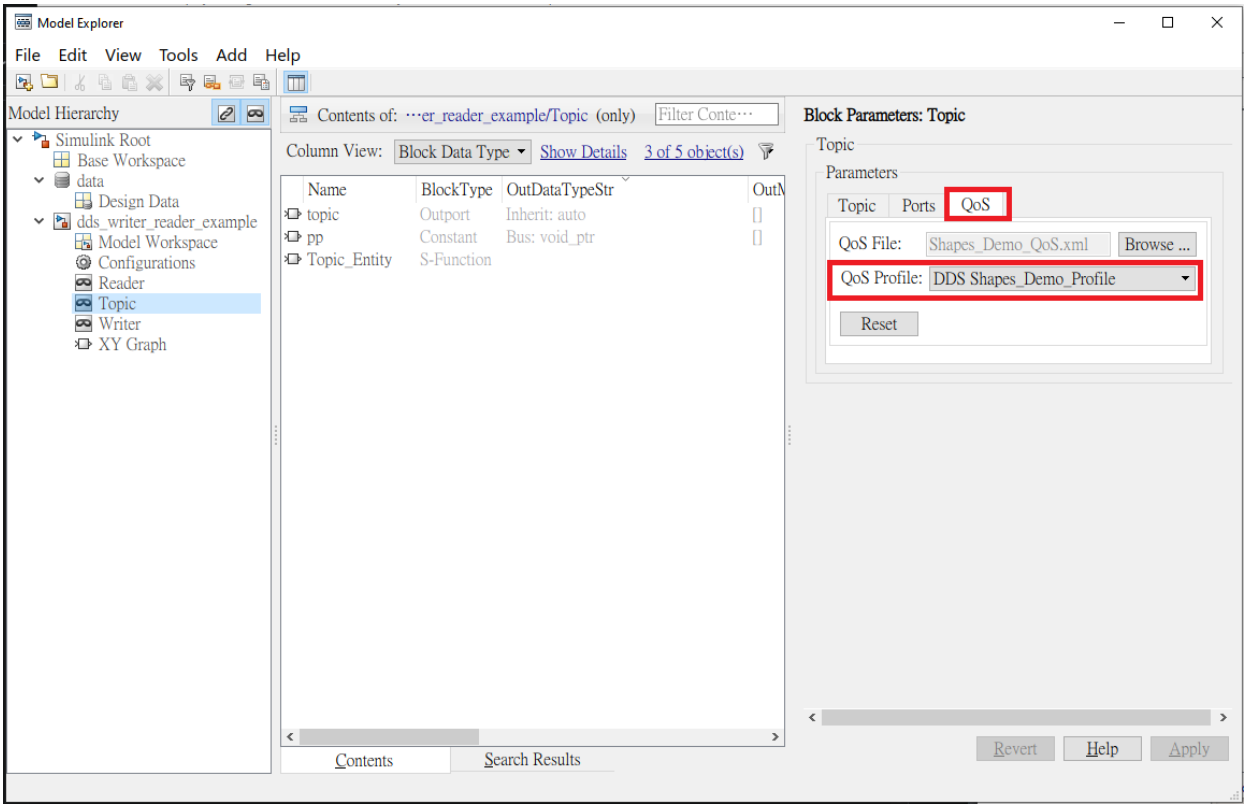
- given an IDLFILENAME, invokes the idlpp tool to generate a MATLAB script. This script is then used to create Simulink Bus objects and import them into the specified data dictionary DCTIONARYFILE. The values are inserted into the 'Design Data' section of the data dictionary.

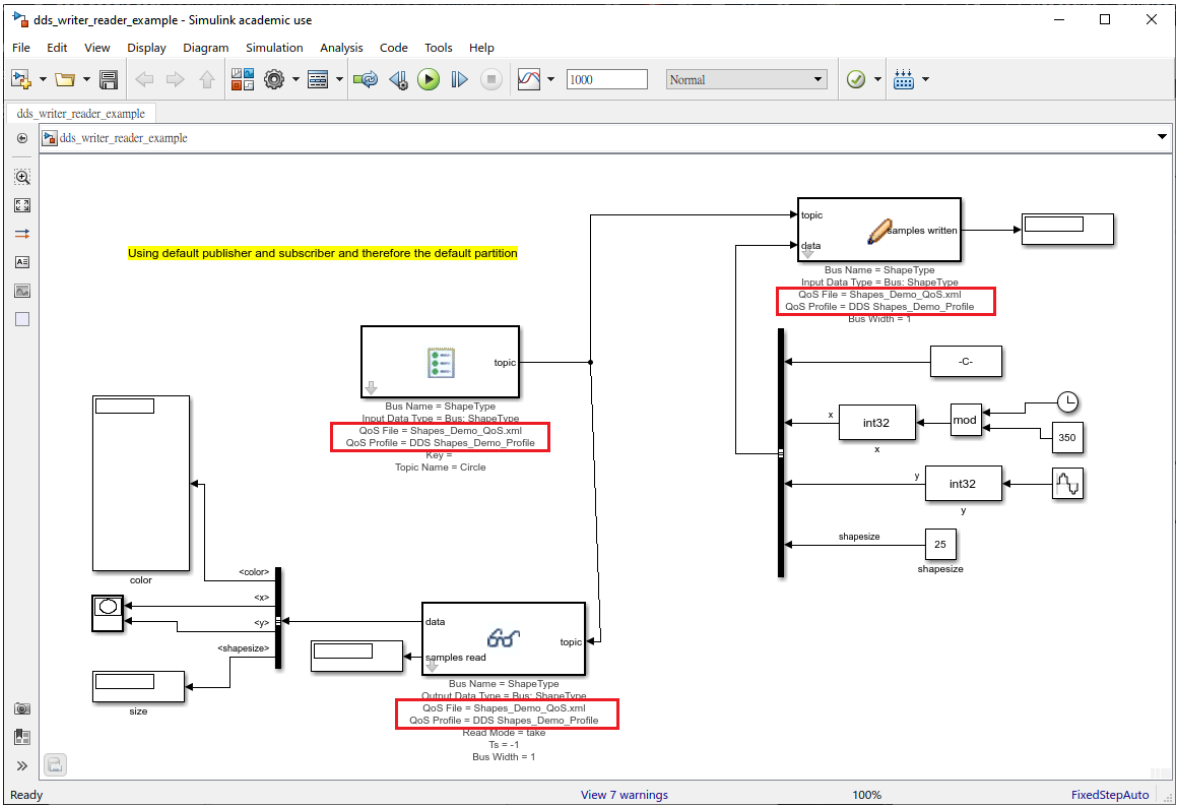
QoS Provider

Each Vortex DDS block has a QoS that can be set using the Block Parameters

Quality of Service for DDS entities is set using XML files based on the XML schema file DDS_QoSProfile.xml

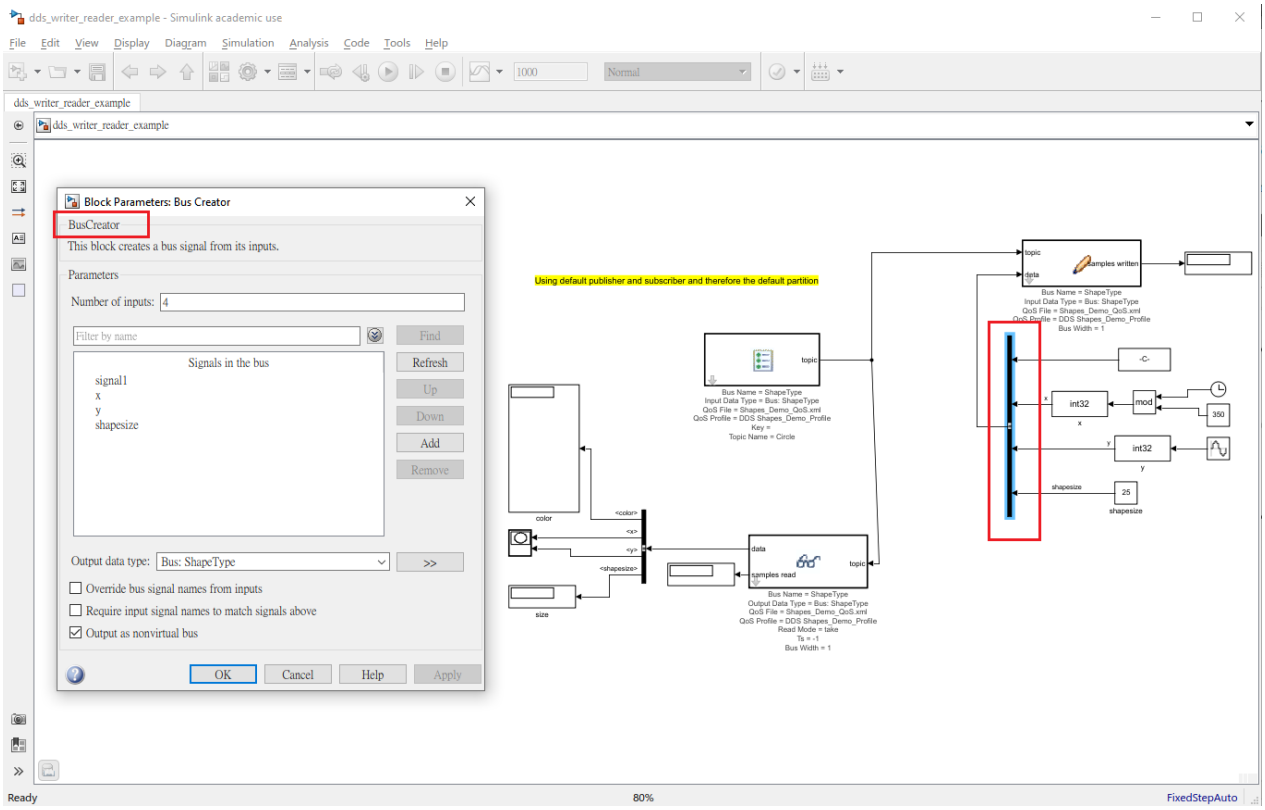
QoS profiles are set using the Simulink block's parameters dialog under the QoS tab. If the QoS File Parameter is set to None the default QoS settings will be used. Seeing the QoS profile in the drop down list only guarantees the QoS profile exists in the file. It does not mean the qos tag exists for the entity. The user is responsible for verifying the neity qos tag exists in the file.


Model Explorer
File Edit View Tools Add Help
Model Hierarchy
Simulink Root
Base Workspace
data
Design Data
dds_writer_reader_example
Model Workspace
Configurations
Reader
Topic
Writer
XY Graph
Contents of: ···er_reader_example/Topic (only)
Filter Conte···
Column View: Block Data Type
Show Details
3 of 5 object(s)
Name
BlockType
OutDataTypeStr
topic
Outport
Inherit: auto
pp
Constant
Bus: void_ptr
Topic_Entity
S-Function
Contents
Search Results
Block Parameters: Topic
Topic
Parameters
Topic
Ports
QoS
QoS File:
Shapes_Demo_QoS.xml
Browse ...
QoS Profile:
DDS Shapes_Demo_Profile
Reset
Revert
Help
Apply

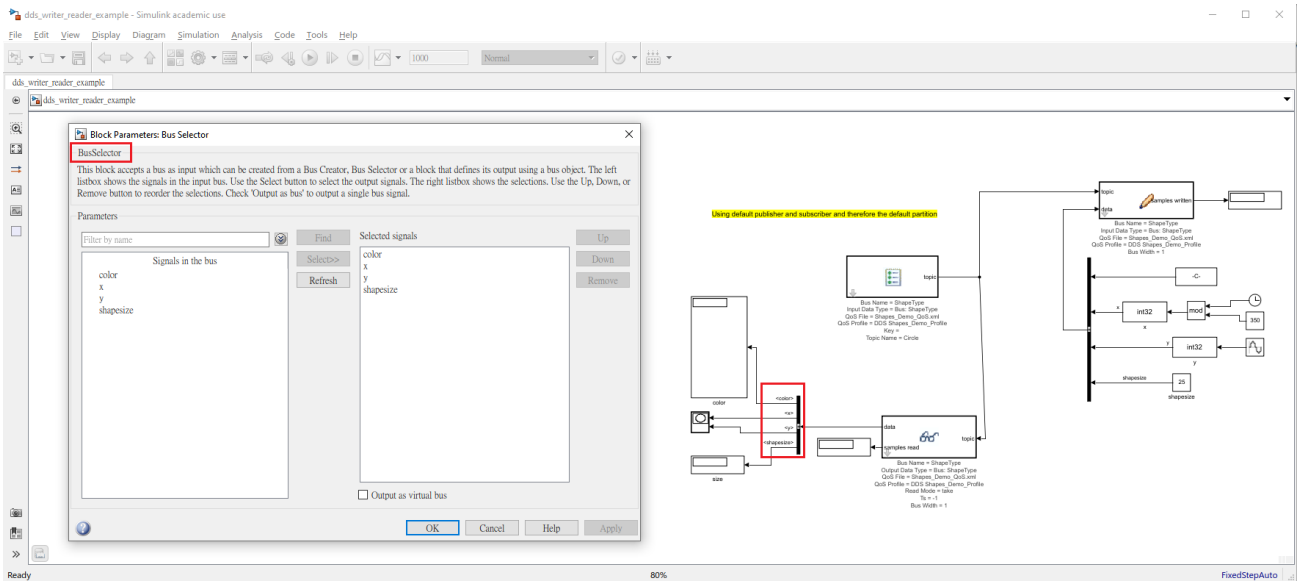

dds_writer_reader_example - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
1000
Normal
dds_writer_reader_example
Using default publisher and subscriber and therefore the default partition
topic
samples written
Bus Name = ShapeType
Input Data Type = Bus: ShapeType
QoS File = Shapes_Demo_QoS.xml
QoS Profile = DDS Shapes_Demo_Profile
Bus Width = 1
-C-
int32
mod
350
int32
shapesize
25
shapesize
Key =
Topic Name = Circle
color
<color>
<x>
<y>
<shapesize>
size
data
topic
samples read
Ts = -1
Ready
View 7 warnings
100%
FixedStepAuto

Within the tutorial, a “Bus Creator” is to be added from the Simulink “Library Browser” under “Simulink/Signal Routing/BusCreator” block.

The BusCreator is used for the DDS Writer block and is responsible for generating sample data.



Within the tutorial, a BusSelector was created for the Reader block to output the received messages to displays.



Topic Block

For a DDS Topic type definition, a corresponding BUS should be defined in the MATLAB workspace. The name of the BUS and the fields and field types should correspond to the DDS topic IDL definition.

In Simulink, a BUS definition can be used as an input or output signal of the Simulink building blocks.

Port Type	Optional	Name	Description	Output consumed by
Input	yes	pp	DDS Domain Participant entity instance	
Output	no	topic	DDS Topic entity instance	Writer, Reader

Domain Block

This block is optional on a DDS Simulink model diagram. If it is not present on a model diagram, a default participant will be created by either a topic, writer or reader block.

In the example, dds_reader_writer_model, no domain participant block is shown.

Port Type	Optional	Name	Description	Output consumed by
Output	no	pp	DDS Domain Participant entity instance	Publisher, Subscriber, Topic

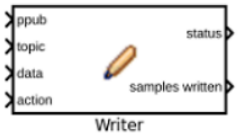
Publisher Block

Port Type	Optional	Name	Description	Output consumed by
Input	yes	pp	DDS Domain Participant entity instance	
Output	no	ppub	DDS Publisher entity instance	Writer

Subscriper Block

Port Type	Optional	Name	Description	Output consumed by
Input	yes	pp	DDS Domain Participant entity instance	
Output	no	psub	DDS Subscriber entity instance	Reader

Writer Block



Port Type	Op-tional	Name	Description	Output consumed by
Input	yes	ppub	DDS Publisher entity instance	
Input	no	topic	DDS Topic entity instance	
Input	no	data	BUS	
Input	yes	action	0 write, 1 dispose, 2 write dispose, 3 no operation	
Output	yes	status	0 for successful writer creation	
Output	yes	samples written	Number of samples written	User

The Data Tab is used to set the input data type (BUS) for the data input port and the bus width (maximum number of samples that can be written per block step). The user must configure source blocks that feed the Writer's data port so that it produces an array of the right size (with valid values for the bus width being any integer >= 1).

The "Reader Available" field in the "Wait for" section is used for specifying if the Writer should wait for the Reader to become available. The associated Timeout field is to specify how long (in seconds) the Writer should wait for the Reader to become available.

Reader Block

Port Type	Optional	Name	Description	Output consumed by
Input	yes	psub	DDS Subscriber entity instance	
Input	no	topic	DDS Topic entity instance	
Output	yes	status	0 for successful reader creation	
Output	no	data	BUS	user
Output	yes	info	BUS	user
Output	yes	samples read	Number of samples read	user

The sample time of a block is a parameter that indicates when, during simulation, the block produces outputs and if appropriate, updates its internal state. Default is -1, meaning it will inherit the Simulink sample time from inputs or the model. Valid values: -1 and Numeric > 0.

"Wait for"

Checking the Historical Data field in the Wait for section specifies that the Reader will wait for historical data to arrive. The Timeout field is for setting time period (in seconds) determining how long the Reader should wait for the historical data. If the timeout is reached, the any remaining historical data may be interleaved with new data.

The "Data available" is for specifying whether the Reader should read only if the data is available. The following Timeouts field determines how long the REader should wait for the availability of data. If the timeout is reached, then the block returns no data and the simulation continues.

5.4 Vortex OpenSplice DDS within Xenomai Kernel 3.1 & Ubuntu 18.04

Running Vortex Opensplice DDS within a Xenomai Kernel in Ubuntu 18.04 consists mainly the following components:

1. Xenomai installed
    a. see Xenomai Installation Guide(see page 65)
2. Building Xenomai application with Vortex OpenSplice DDS(see page 61)

A program/application is strictly running within Xenomai kernel and being managed by Xenomai kernel is when the following conditions are met:

1. The application is running within a patched kernel, which patch changes were made specifically for Xenomai kernel
2. The CPUs which the application is running on are isolated/dedicated for Xenomai (see file "/etc/default/grub")
3. The application has been implemented to execute with Xenomai API's and to be run within Xenomai Cobalt kernel

Work Logs:

- 2020-02-17 Work Log (Installing Xenomai)[71]
- 2020-02-18 Work Log (Booting into Xenomai)[72]
- 2020-02-19 Work Log (Install Xenomai Again)[73]
- 2020-02-20 Work Log (Xenomai Permission Issue)[74]
- 2020-02-21 Work Log (Building Enterprise OpenSplice DDS Evaluation on Xenomai Kernel)[75]
- 2020-02-24 Work Log (Evaluate and Analyze OpenSplice DDS performance within Xenomai Kernel)[76]
- 2020-02-25 Work Log - Achieving Better Latency[77]
- 2020-02-26 Work Log - Latency Analysis[78]
- 2020-03-02 Work Log - Debug Latency Spike[79]
- 2020-03-04 Work Log - Achieving Hard Real-Time[80]
- 2020-03-10 Work Log - Reinstall Xenomai (Downgrade to 3.0.9) for Peak CAN Interface Driver[81]

5.4.1 Building a Xenomai Application with Vortex OpenSplice DDS

Main procedures for making a Xenomai application were taken as follows (see controller_main.cpp[82])

1. Refactor code that needs to be run in Xenomai RT into a function that takes no parameters
2. Create Xenomai tasks that will run the refactored function
    a. one-time task
        i. rt_task_create()
        ii. rt_task_start()
        iii. see "WaitForNodejsRequestRoutine(void*)" in "controller_main.cpp"
    b. periodic tasks

71 https://drive.google.com/open?id=1iKo8Fwf-kVqwVFMCS-pouuLzqH1oxO93LbQlDQolj44
72 https://drive.google.com/open?id=1-K3oSB0vBU5nNN9_Icg9E-x8Bx-JLudOq4zpnEflXuo
73 https://drive.google.com/open?id=18rZiMgpWRWawDViVBXZKNvVBN2LExWsgvI-H-SHQDuk
74 https://drive.google.com/open?id=1K8PBo2ExzPUuVKVJ3PxMr-glMm69LJdmoyGiI0e51A0
75 https://drive.google.com/open?id=1L2BchbJUYPoIODCz8fXDHNOxun9KAn3skTsRDTKDY3I
76 https://drive.google.com/open?id=1oZOHr8x1TgAaj4xD0BiTIYSTBZa5ZCm4wbgt6qJEs-k
77 https://drive.google.com/open?id=1nkmIGCiz4PGUW47Ux6UBvVoihAlgbdYEMkLNTN5NRlA
78 https://drive.google.com/open?id=1rgNMOTIJ0JIkSEtEGDcsG15B30B_PzhxZBsmBsmgkjk
79 https://drive.google.com/open?id=1k1Gtz_yFv14RrLTlToCZ7YlispjTK9Il_MguO0OfurQ
80 https://docs.google.com/document/d/15Faq2Stn72GH-iseAV6DjtMs8n6DIuWAKOdU_Jq5tWM/edit
81 https://docs.google.com/document/d/1DSSUjineL_jjrvEHBa9CjjhJdgE0sH4uqwe0tyS5_0k/edit
82 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/dds-motor-controller/browse/src/motor_control_unit/src/controller_code/controller_main.cpp

        i. rt_task_create()
        ii. rt_task_set_periodic()
        iii. rt_task_start
        iv. see "WriteAndTakeRoutine(void*)" in "controller_main.cpp"
3. Include OpenSplice DDS entities within function
    a. All entities necessary for a communication will need to either be created or made available within the task routine.
    b. these include, but not limited to, the following DDS entities:
        i. DomainParticipant
        ii. Publisher
        iii. Subscriber
        iv. DataWriter
        v. DataReader
        vi. WaitSet
    c. in "controller_main.cpp" a "DDSBridge" class object (see dds_bridge.hpp[83]) is global so that all tasks can use DDS entities in each routine.

Reference:

1. http://www.cs.ru.nl/J.Hooman/DES/XenomaiExercises/

5.4.2 Vortex OpenSplice DDS Community Version on Ubuntu 16.04

2020-01-03 Work Log (Tutorial)[84]

2020-01-06 Work Log (Installation & Tutorial Continued)[85]

2020-01-07 Work Log (HelloWorld & Timestamp)[86]

2020-01-08 Work Log (ContentFiltering & Data-Instances)[87]

2020-01-10 Work Log (DataReader Samples & DataStates)[88]

2020-01-13 Work Log (WaitSets & DataReaders)[89]

2020-01-14 Work Log (Quality of Service, QoS)[90]

2020-01-15 Work Log (Misc)[91]

2020-01-16 Work Log (CMake)[92]

2020-01-17 Work Log (RoundTrip)[93]

---

83 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/dds-motor-controller/browse/src/motor_control_unit/src/dds_code/dds_bridge.hpp
84 https://drive.google.com/open?id=1dkcKNARxRK4pPqUlrHO-IxJvL7_GCj2Wz5NlnpK5mNE
85 https://drive.google.com/open?id=1S_FTthpq-E_kt5JEzkGesoTHvNrM4CL3PaE0_63ROSY
86 https://drive.google.com/open?id=1PXHX3CxX_rXzFojXFiGbUrOMnUWsnECnpDf5rSD7t50
87 https://drive.google.com/open?id=10FhMcAWhesPg6Ifa-jyr5fjV4cOV_6lreFFTNPCloWo
88 https://drive.google.com/open?id=10d98BM9s2DEMvB8_QKK3uMEHtBGveCsticKUpyCxbpw
89 https://drive.google.com/open?id=1QOxSfEz2rlIYpdVfynYSZHWsiV44j8Ocw3FpZqul6zY
90 https://drive.google.com/open?id=11dA5y_L6xbUv1OXuU_W84jRPQa4qcj2H9DWlMvxlDPc
91 https://drive.google.com/open?id=1sZ3xkiqyOxRjb2msV819CQ1wfX36qWBvKAlmYbzaJUo
92 https://drive.google.com/open?id=1NcCEzfIGJz4zemk-8tedJftMZptN_KI0G6L3gX9K0PM
93 https://drive.google.com/open?id=1G5K9vj8C5xHVL7iiXwqeWe-rvmmQ6-rNKydpgnMbhbw

5.4.3 Vortex OpenSplice DDS PDFs

OpenSplice_V5_V6_Migration_Guide[94]

OpenSplice_V4_V5_Migration_Guide[95]

OpenSplice_Tutorial_C[96]

OpenSplice_TunerGuide[97]

OpenSplice_TesterUserGuide[98]

OpenSplice_StreamsAPIReference[99]

OpenSplice_SecureNetworkingGuide[100]

OpenSplice_RnRManagerGuide[101]

OpenSplice_RnRAPIReference[102]

OpenSplice_RMIUserGuide[103]

OpenSplice_refman_Java[104]

OpenSplice_refman_CPP[105]

OpenSplice_refman_C[106]

OpenSplice_PythonDCPSAPIGuide[107]

OpenSplice_NodeMonitorGuide[108]

OpenSplice_NodeJSDCPSAPIGuide[109]

OpenSplice_ModelingGuide[110]

OpenSplice_IDLPreProcGuide[111]

OpenSplice_GPBTutorial[112]

OpenSplice_GettingStartedGuide[113]

---

94 https://drive.google.com/open?id=1NJyaa9JRAJgUbHbhkRv1NGD68tPrwVcF
95 https://drive.google.com/open?id=1MazvrBmSHJPqS0wkSIUmaXxFhmDw3V64
96 https://drive.google.com/open?id=1CEizwswawJxRCzUfrhV15fT7cwej1VAV
97 https://drive.google.com/open?id=1R5Hz_OmM7CKQIkIYvmZ5DA0qoPYd76v9
98 https://drive.google.com/open?id=199asdh_KVYdQ4EKMZR1KFzSzOL8fNlqH
99 https://drive.google.com/open?id=1An-2fNqKGV7o_WeHTqIoyOU_9L3aPY7U
100 https://drive.google.com/open?id=1juT5qGCk0eihUgx0_XnFazwOBTAKYYAl
101 https://drive.google.com/open?id=1phksQ6RXPLN5148fGRyaBoRv1_K9R42t
102 https://drive.google.com/open?id=1H1S65fJH9FXovkhjB2NHJeDNqdYsIkS0
103 https://drive.google.com/open?id=13DegO2y4KijTE3APpD0kJ0tW8sOsRqBL
104 https://drive.google.com/open?id=1FU7bte44iM21lJKk6i5bzLNO4-mdUyy9
105 https://drive.google.com/open?id=16-Su4_UZ6wTXbBHLcGMLnQ96CEQgfqyS
106 https://drive.google.com/open?id=1MAhinDMj2OuiLXQ9S8nkUR37S__cCBgG
107 https://drive.google.com/open?id=1BM5pPjuFf_BjJMvBDD3ahjQsmfSSCC6E
108 https://drive.google.com/open?id=1Tlf2kVGoMqiI9Tb_y3pkHemytyV5_BGY
109 https://drive.google.com/open?id=1XTlzNwelNOL7x6a0mHGBRuw23IMPyIav
110 https://drive.google.com/open?id=1gGLsxSCGwh2-5EU3jlp7h3d8kl_LhpNq
111 https://drive.google.com/open?id=1NWEW0bwzezGTqDK2y85YZJ6R8Bx1uxYV
112 https://drive.google.com/open?id=1eslVkHKpbYckZuzeJa8s2yKNbPBq9RDm
113 https://drive.google.com/open?id=1qBEue5Fmjxyy6jG32eg5adMuea-smQKL

OpenSplice_EvaluationGuide[114]

OpenSplice_DeploymentGuide[115]

OpenSplice_DDSTutorial[116]

OpenSplice_DDSSimulinkGuide[117]

OpenSplice_DDSMATLABGuide[118]

OpenSplice_DDSLabVIEWGuide[119]

114 https://drive.google.com/open?id=1tGbqLoznrOCdIW_gWWzIZfC7onOtnYCD
115 https://drive.google.com/open?id=1UeW2UjMG_mldABJCM812Upj_k_veQuSf
116 https://drive.google.com/open?id=1dSaErEqsezfyUEUdGFM4jrsndNvNFKJQ
117 https://drive.google.com/open?id=1MztEc3IJM9zRNPLSHe3pdju0zyoBvWdp
118 https://drive.google.com/open?id=17G979PlqOAVmUWY7mq9wimnYHBRMcpQV
119 https://drive.google.com/open?id=1EgYJWA9wo75-mroqlM2oSBsl81XWIy0P

6 Xenomai Installation Guide

6.1 Installing and Running Xenomai on the Custom-Built Linux Kernel

1. Allowing non-root users to use Xenomai
    a. Granting privileges to other users to run applications within Xenomai kernel
        i. adding a new group with name "xenomai" and gid=1234
            1. `sudo addgroup xenomai --gid 1234`
            2. note: your "/etc/default/grub" should have GRUB_CMDLINE_LINUX_DEFAULT="xenomai.allowed_group=1234" from specifying the kernel parameter when booting into Xenomai kernel
        ii. adding root privilege to xenomai group
            1. `sudo addgroup root xenomai`
            2. note: check with `groups root`. "xenomai" should be in the output to show that user "root" is in group "xenomai".
        iii. adding current user to the group xenomai
            1. `sudo usermod -a -G xenomai $USER`
            2. note: check with `groups $USER`. "xenomai" should be in the output to show that user "$USER" is in group "xenomai". $USER is probably "d400" in most machines.
        iv. reboot to let settings take effect
2. Installing Xenomai-3.0.9 Libraries and Tools
    a. With the downloaded Xenomai code from Patching, Building, and Booting into a Xenomai-Compatible Linux Kernel(see page 67) (which we have only used the script file for the ipipe patch file so far), we can compile and install Xenomai-3.0.9 libraries and tools
        i. enter into Xenomai-3.0.9 directory
        ii. run `./configure --with-core=cobalt --enable-smp --enable-pshared --enable-registry`
            1. note: don't include "--enable-registry" in a virtual machine
            2. would need the following packages from apt-get
                a. "pkg-config"
                b. "libfuse-dev"
        iii. `make -j8`
            1. if encountered an error of "READ_ONCE" from file "urw.h" then do the following
                a. open file editor for file "[xenomai_directory]/include/cobalt/uapi/kernel/urw.h"
                b. search for "READ_ONCE"
                c. replace it for "ACCESS_ONCE"
        iv. `sudo make install`
            1. by default this will install xenomai libraries and tools at directory "/usr/xenomai"
    b. Check if installation succeeded
        i. `ls -lath /usr` and `ls -lath /usr/xenomai` to check if the directory along with the libraries & tools were created
    c. Take care of environment variables
        i. edit file "~/.bashrc"
        ii. append at the end of the file two lines:
            1. "export XENOMAI_ROOT_DIR=/usr/xenomai"
            2. "export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/xenomai/lib"
        iii. source it
            1. `source ~/.bashrc` or `. ~/.bashrc`
3. Run Xenomai Latency test

a. ~~add Xenomai's binary directory to environment variable by running~~
    i. ~~going to Xenomai's directory~~
    ii. ~~`source xenomai_source.sh`~~
        1. ~~automatically loads this environment variable by appending the following line in the end of file "~/.bashrc"~~
            a. ~~"source [xenomai_directory_path]/xenomai-3.0.9/xenomai_source.sh"~~
            b. ~~e.g.: "source /home/d400/xenomai-3.0.9/xenomai_source.sh"~~
    iii. ~~The above source command will add Xenomai's "bin" and "sbin" directories to $PATH~~
        1. ~~can check by `echo $PATH` and seeing if "/usr/xenomai/bin" and "/usr/xenomai/sbin" exist~~
b. Run "latency"
    i. `/usr/xenomai/bin/latency`
        1. if encountered error of "cannot open RTDM device /dev/rtdm/memdev-private: Permission denied" when running with regular user, do the following:
            a. `sudo vim /etc/udev/rules.d/99-xenomai-real-time-devices.rules`
            b. add a line KERNEL=="memdev-private",SUBSYSTEM=="rtdm",MODE="0660",GROUP="xenomai"
                i. this line of command (will automatically run during system startup) will find a device that match keys of KERNEL and SUBSYSTEM (these two keywords are each associated with two equal signs "==")
                ii. and then sets the mode to 0660 to give read & write access to the current user and group
                iii. and then sets the group of the device to "xenomai"
            c. reboot machine to let the rtdm device be reconnected in order to apply this newly created rules
            d. verify with `ls -lath /dev/rtdm` after rebooting, if the groups for "memdev-private" and "memdev-shared" are of group "xenomai" then the rules have been applied successfully.
    ii. if it runs successfully it means Xenomai has been installed successfully
c. Running "xeno-test"
    i. `/usr/xenomai/bin/xeno-test`
        1. if countered error similar to "sysregd: create_directory_recursive("/var/run/xenomai/d400/anon@2244"): Permission denied", then do the following:
            a. `sudo mkdir /var/run/xenomai`
            b. `sudo chgrp xenomai /var/run/xenomai`
            c. `sudo chmod g+rw /var/run/xenomai`
        2. (Optional) If RTnet is enabled, then insert the following kernel modules to avoid modprobe errors (such as "modprobe: ERROR: could not insert '[kernel module name]': Operation not permitted"):
            a. `sudo insmod /lib/modules/4.4.182-xenomai-3.0.9/kernel/drivers/xenomai/net/drivers/rt_loopback.ko`
            b. `sudo insmod /lib/modules/4.4.182-xenomai-3.0.9/kernel/drivers/xenomai/net/stack/ipv4/rtipv4.ko`
            c. `sudo insmod /lib/modules/4.4.182-xenomai-3.0.9/kernel/drivers/xenomai/net/stack/packet/rtpacket.ko`

Next stop: Xenomai Tutorial - Creating a Simple Program for Xenomai(see page 40)

6.2 Patching, Building, and Booting into a Xenomai-Compatible Linux Kernel

Xenomai is a bunch of stuffs put together to transform a regular vanilla Linux Kernel into a Real-Time one. These stuffs include mainly:

1. Some modifications to the vanilla Linux Kernel to make it a real-time one
    a. this is called "patching the kernel" or "updating/modifying the kernel with a patch"
2. Some API (application programming interface) to let the user access the real-time functionalities within the modified Linux Kernel

The following procedures should allow a user to install Xenomai (version 3.0.9) on a Linux Kernel (version 4.4.182) that has been patched with ipipe-core (version 4.4.182-x86-15):

6.2.1 Downloading the Ipipe Patch file with Compatible Linux Kernel

1. Xenomai has a list of compatible Linux Kernel that can be successfully patched. See: https://xenomai.org/downloads/ipipe/v4.x/x86/
    a. In our build, "ipipe-core-4.4.182-x86-15.patch" was downloaded
        i. The patch file is called ipipe-core
        ii. The patch file is for patching a kernel with version 4.4.182
        iii. The patch file is for CPU system architecture x86
        iv. The patch file is the 15th revision
2. You can check your current version of Linux Kernel by `uname -r`

6.2.2 Downloading a Xenomai-Compatible Linux Kernel Source Tree

1. Linux Kernel Source Tree = Linux Kernel Source Code
2. Head to the official Linux archive of Linux Kernel source codes:
    a. https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/refs/tags?h=linux-4.19.y
3. Search for the Linux Kernel version that matches according to the ipipe patch file downloaded in step 1
    a. In our build, "v4.4.182" was downloaded
4. Extract/unzip the Linux Kernel source tree from "linux-4.4.182.tar.gz" and put the source tree in your home directory
    a. home directory is "~/"
        i. `echo ~`
        ii. `echo $HOME`

6.2.3 Patching the Linux Kernel

1. A script file within Xenomai's API is required in order to patch the kernel with the downloaded ipipe file. We will install the Xenomai API and use it to patch the Linux Kernel:
    a. Download Xenomai-3.0.9 at https://xenomai.org/downloads/xenomai/stable/
    b. (Optional) put the download ipipe patch file under "xenomai-3.0.9/scripts"
    c. locate a script file "scripts/prepare-kernel.sh"
    d. run it
        i. `prepare-kernel.sh --linux=<linux-srctree> --ipipe=<ipipe-patch> --arch=<target-arch>`
        ii. e.g.:

1. `/home/d400/xenomai-3.0.9/scripts/prepare-kernel.sh --linux=/home/d400/linux-4.4.182 --arch=x86 --ipipe=/home/d400/xenomai-3.0.9/scripts/ipipe-core-4.4.182-x86-15.patch`
    a. The absolute path "/home/d400/xenomai-3.0.9/scripts/prepare-kernel.sh" is used
    b. The absolute path of our downloaded Linux Kernel Source Tree "/home/d400/linux-4.4.182" is used here
    c. The architecture "x86" is specified
    d. The absolute path of our downloaded ipipe patch file "/home/d400/xenomai-3.0.9/scripts/ipipe-core-4.4.182-x86-15.patch" is used
e. Note: Xenomai has two kinds of installation, mercury and cobalt. We are installing cobalt (co-kernel). For more detailed info see https://gitlab.denx.de/Xenomai/xenomai/-/wikis/Start_Here
2. The Linux kernel source code is patched after successfully running the script file and will be ready for the next step
3. Note: Xenomai will only work with ipipe's that are released prior to Xenomai's release.

6.2.4 Configuring the Linux Kernel

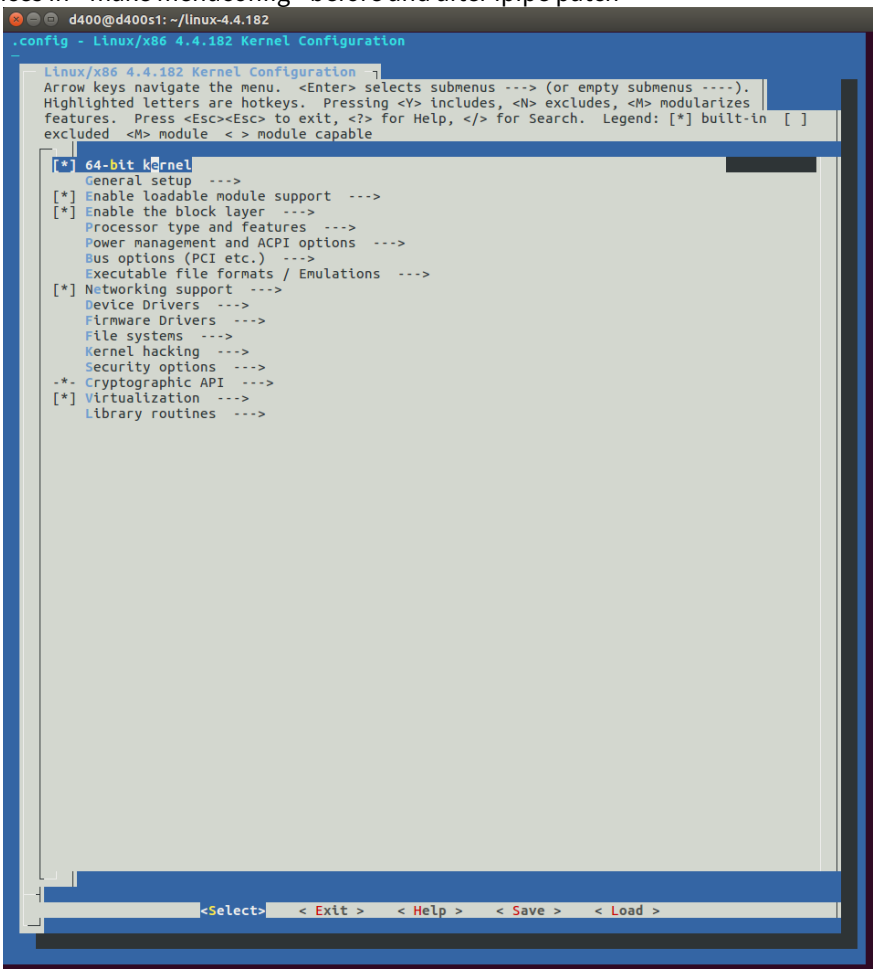
1. There are quite some options we need to configure before compiling/building the Linux kernel.
    a. enter into the Linux Kernel source tree directory that you have extracted under home directory
    b. (Optional) make a default .config file
        i. `make ARCH=x86_64 defconfig`
    c. Install packages needed to build a kernel
        i. `sudo apt-get install kernel-package`
        ii. see https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel for more info
    d. `make menuconfig` to for kernel configuration
        i. (Debug) If menuconfig has error about can't finding file "arch/$SRCARCH/xenomai/Kconfig" then simply go to file "init/Kconfig" at line 2056 and replace $SRCARCH with x86 (or the architecture of your machine)
        ii. In particular, Xenomai requires the following kernel options to be set within menuconfig:
            1. → General setup
                a. → Local version - append to kernel release = -xenomai-3.0.9
                    i. IMPORTANT!!! This configuration is important for later to distinguish our newly built kernel from the old one
                    ii. without specifying it, the built kernel would be named "linux-4.4.182"; with it specified, name would be "linux-4.4.182-xenomai-3.0.9"
                a. → Timers subsystem
                    i. → High Resolution Timer Support (Enable)
            2. → Xenomai/cobalt
                a. → Sizes and static limits
                    i. → Number of registry slots (512 → 4096)
                    ii. → Size of system heap (Kb) (512 → 4096)
                    iii. → Size of private heap (Kb) (512 → 4096)
                    iv. → Size of shared heap (Kb) (64 → 256)
                    v. → Maximum number of POSIX timers per process (128 → 512)
                b. → Drivers
                    i. → RTnet
                        i. → RTnet, TCP/IP socket interface (Enable)
                            i. → Drivers
                                i. → New intel(R) PRO/1000 PCIe (Enable)
                                ii. → Realtek 8169 (Enable)
                                iii. → Loopback (Enable)

                ii. → Add-Ons
                    i. → Real-Time Capturing Support (Enable)
            ii. → CAN drivers
                i. → RT-Socket-CAN, CAN raw socket interface (Enable)
    3. → Power management and ACPI options
        a. → CPU Frequency scaling
            i. → CPU Frequency scaling (Disable)
                i. this might need "Multi-core scheduler support" to be disabled first
        b. ACPI (Advanced Configuration and Power Interface) Support
            i. Processor (Disable)
        c. CPU Idle
            i. CPU idle PM support (Disable)
                i. might output warning message, but ok to ignoreN
    4. → Processor type and features
        a. → Enable maximum number of SMP processors and NUMA nodes (Disable)
            i. may not be shown because it's automatically disabled, can search for "CONFIG_MAXSMP" with "/" inside menuconfig
        b. → Processor family
            i. → Core 2/newer Xeon
        c. → Multi-core scheduler support (Disable)
        d. → Transparent Hugepage support (Disable)
        e. → Allow for memory compaction (Disable)
        f. → Contiguous Memory Allocation (Disable)
        g. → Page Migration (Disable)
            i. note: Page Migration can only be disabled after disabling "Transparent Hugepage support", "Allow for memory compaction", and "Contiguous Memory Allocation" because it depends on them.
    5. → Device Drivers
        a. → Staging drivers
            i. → Unisys SPAR driver support
                i. → Unisys visorbus driver (Disable)

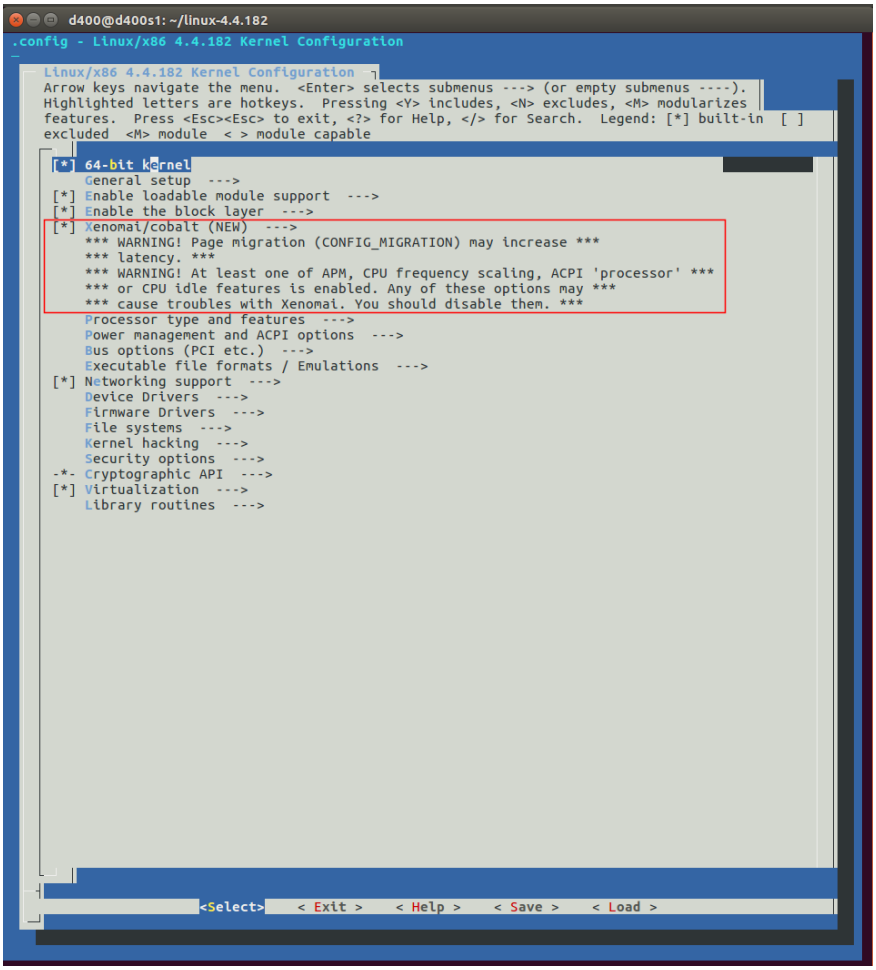
iii. Notes:
    1. setting for each
        a. [ ] = exclude
        b. [*] = include/built-in
        c. [M] = module
    2. in menuconfig, when an option cannot be excluded, it means there are other options that depends on this. It is after the other depending options were excluded that the current one can be deselected.
    3. it can be useful to use "/" for finding with keywords
    4. it can be useful to use "?" or "h" to see which other options are dependent on the current one or just to check its description
    5. to remove a compiled kernel remove the following files with commands (replace "KERNEL_VERSION" with your kernel version):
        a. `rm -rf /boot/vmlinuz*KERNEL_VERSION*`
        b. `rm -rf /boot/initrd*KERNEL_VERSION*`
        c. `rm -rf /boot/System-map*KERNEL_VERSION*`
        d. `rm -rf /boot/config-*KERNEL_VERSION*`
        e. `rm -rf /lib/modules/*KERNEL_VERSION*`
        f. `rm -rf /var/lib/initramfs/*KERNEL_VERSION*`
        g. `sudo update-grub`

h. see reference: commands in "hard way" of https://askubuntu.com/questions/594443/how-can-i-remove-compiled-kernel

6. differences in `make menuconfig` before and after ipipe patch

a. 

b.



iv. Reference:
    1. https://rtt-lwr.readthedocs.io/en/latest/rtpc/xenomai3.html

6.2.5 Compiling the Linux Kernel

After the configuration, we can compile/build the Linux kernel. This may take a long time depending on your CPU cores.

1. Compile/Build the kernel
    a. `sudo make -j[number of threads]`
        i. for example `sudo make -j4`
    b. Note: this process can take up to anywhere from 10-20 minutes to 1-2 hours to finish
2. Generate modules dependency and map files after installing kernel and kernel modules
    a. `sudo depmod -a`
3. Install kernel modules
    a. `sudo make modules_install`
4. Install the kernel
    a. `sudo make install`
    b. the new kernel should be installed in your "/boot" directory with some kernel modules under "/lib/modules". You can use `sudo find / "*xenomai*"` to check it

6.2.6 Booting into the Newly Built Linux Kernel with GRUB bootloader

To boot into another Linux Kernel, we would need to find the kernel's associated menuentry (like an identification for the kernel), then modify the GRUB bootloader's (GRUB = GRand Unified Bootloader) configuration file, and lastly update GRUB according to the configuration file to take effect after a restart:

1. WARNING! Make sure you have a way to physically reboot the computer before proceeding!
    a. if the kernel did not boot successfully (which can occur quite often), a physical machine reboot would be required to reset (ssh would not be available to issue `reboot` command). Even if we can set to boot into a kernel only on next reboot (which we would do in a second), a physical reboot would still be required when the boot loader encounters boot failure.
2. To boot into a different kernel ONLY on next reboot
    a. find the kernel's menuentry in "/boot/grub/grub.cfg" to be used to specify which kernel to boot into
        i. `grep -rni "menuentry" /boot/grub/grub.cfg`

```
d400@u16-01-RTComputer:/$ grep -nri "menuentry" /boot/grub/grub.cfg
22:if [ x"${feature_menuentry_id}" = xy ]; then
23:  menuentry_id_option="--id"
25:  menuentry_id_option=""
28:export menuentry_id_option
133:menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnul
inux-simple-7d2faa1b-bbda-40f2-ae72-6e154e484246' {
150:submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-7d2faa1b-bbda-40f2-ae72
-6e154e484246' {
151:    menuentry 'Ubuntu, with Linux 4.19.59-rt24' --class ubuntu --class gnu-linux --class gnu --class
os $menuentry_id_option 'gnulinux-4.19.59-rt24-advanced-7d2faa1b-bbda-40f2-ae72-6e154e484246' {
170:    menuentry 'Ubuntu, with Linux 4.19.59-rt24 (recovery mode)' --class ubuntu --class gnu-linux --cl
ass gnu --class os $menuentry_id_option 'gnulinux-4.19.59-rt24-recovery-7d2faa1b-bbda-40f2-ae72-6e154e484
246' {
187:    menuentry 'Ubuntu, with Linux 4.15.0-58-generic' --class ubuntu --class gnu-linux --class gnu --c
lass os $menuentry_id_option 'gnulinux-4.15.0-58-generic-advanced-7d2faa1b-bbda-40f2-ae72-6e154e484246' {
206:    menuentry 'Ubuntu, with Linux 4.15.0-58-generic (recovery mode)' --class ubuntu --class gnu-linux
 --class gnu --class os $menuentry_id_option 'gnulinux-4.15.0-58-generic-recovery-7d2faa1b-bbda-40f2-ae72
-6e154e484246' {
223:    menuentry 'Ubuntu, with Linux 4.4.182-xenomai-3.0.9' --class ubuntu --class gnu-linux --class gnu
 --class os $menuentry_id_option 'gnulinux-4.4.182-xenomai-3.0.9-advanced-7d2faa1b-bbda-40f2-ae72-6e154e4
84246' {
242:    menuentry 'Ubuntu, with Linux 4.4.182-xenomai-3.0.9 (recovery mode)' --class ubuntu --class gnu-l
inux --class gnu --class os $menuentry_id_option 'gnulinux-4.4.182-xenomai-3.0.9-recovery-7d2faa1b-bbda-4
0f2-ae72-6e154e484246' {
259:    menuentry 'Ubuntu, with Linux 4.4.182-xenomai-3.0.9.old' --class ubuntu --class gnu-linux --class
 gnu --class os $menuentry_id_option 'gnulinux-4.4.182-xenomai-3.0.9.old-advanced-7d2faa1b-bbda-40f2-ae72
-6e154e484246' {
278:    menuentry 'Ubuntu, with Linux 4.4.182-xenomai-3.0.9.old (recovery mode)' --class ubuntu --class g
nu-linux --class gnu --class os $menuentry_id_option 'gnulinux-4.4.182-xenomai-3.0.9.old-recovery-7d2faa1
b-bbda-40f2-ae72-6e154e484246' {
307:menuentry 'Windows Boot Manager (on /dev/nvme1n1p1)' --class windows --class os $menuentry_id_option
'osprober-efi-A9EA-9D34' {
325:menuentry 'System setup' $menuentry_id_option 'uefi-firmware' {
d400@u16-01-RTComputer:/$
```

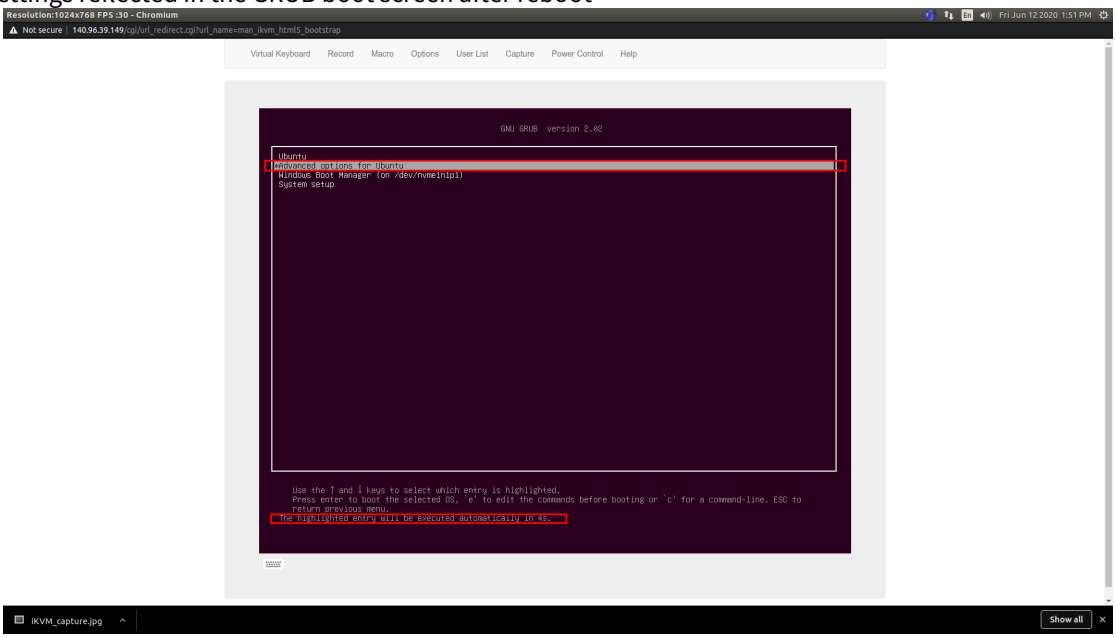
        1.
        2. look for "submenu" and "menuentry" in this search. If you would enable bootloader screen to manually select a kernel to boot into, the items of "submenu" and "menuentry" would appear there. But for a user booting into a new kernel via ssh, the bootloader screen option would not be viable because ssh would only start after booting into a kernel. Therefore, manually choosing which kernel to boot into before restart is a must for that use-case.
    b. use `grub-reboot` to choose which kernel to boot into for the next restart
        i. e.g.: `sudo grub-reboot "Advanced options for Ubuntu>Ubuntu, with Linux 4.4.182-xenomai-3.0.9"`
    c. reboot machine
    d. if system successfully reboot, verify that we have booted into the newly built xenomai kernel (with previously kernel configuration to append "-xenomai-3.0.9" at the end of the kernel name)
        i. `uname -r` to check for current linux kernel name, should be exactly "4.4.182-xenomai-3.0.9"
            1. if the name is not that, that means we have booted into an undesired linux kernel. Check again the menuentry used for `grub-reboot` and rerun it.
                a. to check previous commands, run `history` or `history | grep grub`
    e. run `dmesg | grep -i xenomai` to verify that xenomai has started successfully
    a. should show "[Xenomai] Cobalt v3.0.9" if Xenomai is booted correctly

b.
```
d400@d400s1:~$ dmesg | grep -i xenomai
[    0.000000] Linux version 4.4.182-xenomai-3.0.9 (root@d400s1) (gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu
1~18.04) ) #1 SMP Thu Jun 11 17:13:18 CST 2020
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.4.182-xenomai-3.0.9 root=UUID=3b1904c9-3f50-4f0e-
ab04-f398a473d098 ro quiet splash
[    0.000000] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-4.4.182-xenomai-3.0.9 root=UUID=3b1904c9-3f5
0-4f0e-ab04-f398a473d098 ro quiet splash
[    3.894009] [Xenomai] scheduling class idle registered.
[    3.894010] [Xenomai] scheduling class rt registered.
[    3.894101] I-pipe: head domain Xenomai registered.
[    3.895159] [Xenomai] Cobalt v3.0.9
d400@d400s1:~$
```

6.2.7 Permanently Booting into Xenomai Kernel

If the correct kernel has been booted into and Xenomai has started successfully, we can choose to permanently boot into the newly built xenomai kernel. This would involve modifying the grub configuration file "/etc/default/grub" and specifying some kernel configurations:

1. (Optional) To always show GRUB menu for 15 seconds
    a. "GRUB_TIMEOUT_STYLE=false"
    b. "GRUB_TIMEOUT=15"
2. To set the default Linux Kernel for GRUB to boot into
    a. GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux 4.4.182-xenomai-3.0.9"
        i. don't forget the quotation marks
3. Exit file /etc/default/grub and run `sudo update-grub` with new configuration
4. Reboot
5. Once at the GRUB menu, make sure "Advanced options for Ubuntu" is highlighted.
    a. if it's not, that means something is wrong with "GRUB_DEFAULT" in your /etc/default/grub file
6. Note: settings reflected in the GRUB boot screen after reboot

a.


    i. this is the default selection from GRUB, without any keyboard input from the physical machine before booting into a kernel

6.2.8 **Kernel Parameters for Xenomai**

Xenomai allow dedicated resources such as dedicated CPU cores to be used exclusively for Xenomai related functionalities (similar to creating a new virtual machine with the number of CPU cores to allocate it). More so, only particular groups specified by Xenomai can access these resources (see File Permission[120]). These would be the particular parameters to configure in "/etc/default/grub" file:

1. To allow a specific group to use Xenomai
    a. GRUB_CMDLINE_LINUX_DEFAULT="xenomai.allowed_group=1234"
2. To set the CPUs that Xenomai will be using
    a. GRUB_CMDLINE_LINUX_DEFAULT="xenomai.supported_cpus=0x000000F0"
        i. take consideration on how many CPU cores you have (can use `htop` to check)

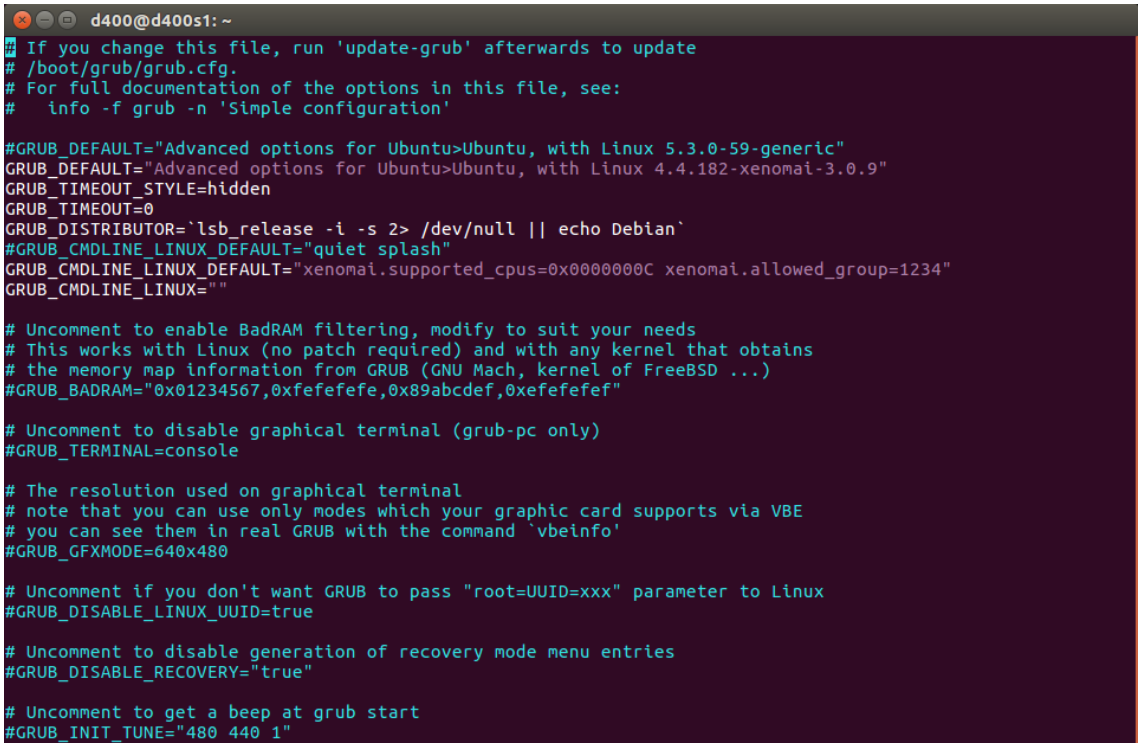
1. ~~To isolate CPUs~~
    a. ~~GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=4,5,6,7"~~
    b. ~~note: to detect if cpus are isolated check file "/sys/devices/system/cpu/isolated" after reboot~~
    c. ~~reference: how to detect if isolcpus is activated?[121]~~

- ~~To force grub to use acpi~~

1. ~~GRUB_CMDLINE_LINUX_DEFAULT="acpi=force"~~

- (Optional) To not have irq service balance load on CPUs
    a. GRUB_CMDLINE_LINUX_DEFAULT="acpi_irq_nobalance"
- Putting it all together
    a. GRUB_CMDLINE_LINUX_DEFAULT="xenomai.supported_cpus=0x000000F0 xenomai.allowed_group=1234 acpi_irq_nobalance"
- quit file /etc/default/grub and update GRUB with `sudo update-grub`
- reboot
- check to see if xenomai has dedicated CPU cores
    a. with xenomai's configuration file
        i. `cat /proc/xenomai/affinity`
            1. this file should have the same affinity value as "xenomai.supported_cpus"
    b. with `stress` and observe with `htop`
        i. run `stress --cpu [number of total cpu]`
        ii. run `htop` on another terminal to observe
            1. the dedicated CPU cores should have no CPU usages while the others should be fully loaded
            2. e.g.: If you have 8 CPU cores and you specify "xenomai.supported_cpus" to be "0x000000F0", then CPUs #4, #5, #6, #7 (or CPUs #5, #6, #7, #8) will be dedicated for Xenomai. If `stress --cpu 8` is runned, those dedicated CPUs should not have any CPU usage.
                a. (Debug) If the supposedly dedicated CPU cores shows CPU usages, then that means Xenomai has not successfully dedicated the CPU cores. Check for spelling errors in specifying the kernel parameters and make sure to `sudo update-grub` before reboot.
- An example file "/etc/default/grub"

120 http://mmslswdev.itri.org.tw/confluence/display/TRAIN/File+Permission
121 https://unix.stackexchange.com/questions/336017/how-to-detect-if-isolcpus-is-activated

a.


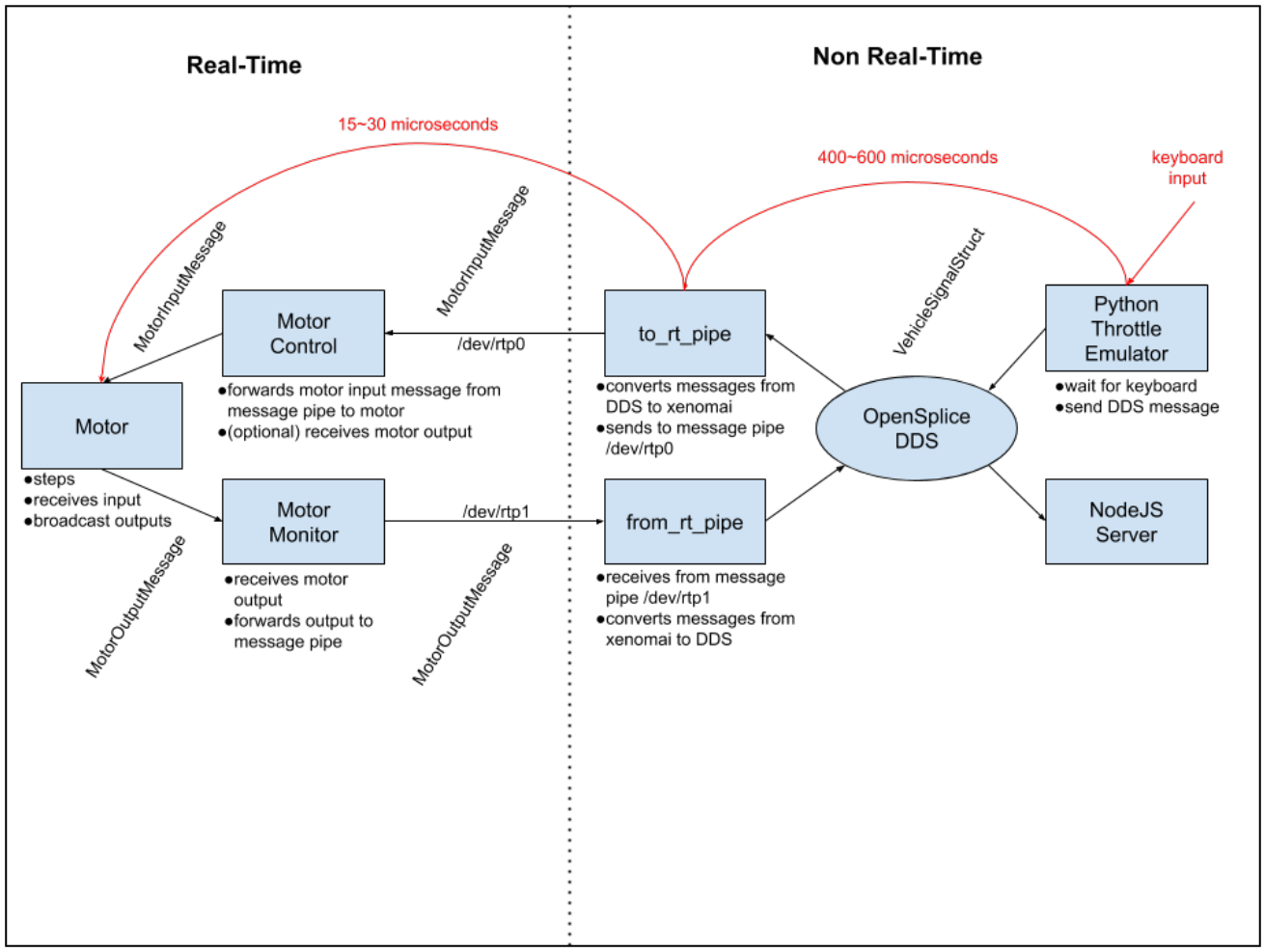
- ~~Note: you can check "/usr/src/linux-headers-4.19.89-xenomai-3.0.9/.config" to see whether configurations from menuconfig has taken effect in the installed kernel~~

6.2.9 **FINISHED!**

You have successfully built a Xenomai-compatible Linux kernel with Xenomai Cobalt kernel running on the CPUs specified by "xenomai.supported_cpus"! Now to install Xenomai library and run some pre-built latency tests with Installing and Running Xenomai on the Custom-Built Linux Kernel(see page 65)

7 Python to Xenomai RT Demo

Current Software Design:



The demo on commit cc5e00746a5[122] for the AVDS Platform includes the following nodes with their tasks:

1. A Python throttle emulator[123]
   a. receives keyboard input and sends DDS message (VehicleSignalStruct)
2. Pipes between non real-time and real-time applications (to_rt_pipe[124] and from_rt_pipe[125])
   a. sends message from non real-time to real-time and vice versa
   b. converts DDS message (VehicleSignalStruct) to Xenomai (MotorInputMessage/MotorOutputMessage) messages and vice versa
3. Motor[126]

122 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/commits
123 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/python_code/throttle_emulator.py
124 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/dds_code/to_rt_pipe.cpp
125 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/dds_code/from_rt_pipe.cpp
126 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/model_code/ert_main.cpp

4. Motor Control[127] for sending Motor input commands
5. Motor Monitor[128] to receive Motor output

Working Demo from Python to Motor:

- Top Left = Motor
- Top Right = Motor Control
- Bottom Right = to_rt_pipe (start OpenSplice DDS)
- Bottom Left = Python Throttle Emulator

Working Demo with Long Key Press:

127 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/controller_code/controller_main.cpp

128 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/src/motor_control_unit/src/xenomai_code/motor_monitor.cpp


python-to-motor-long-press.mp4

8 RT Modular Node

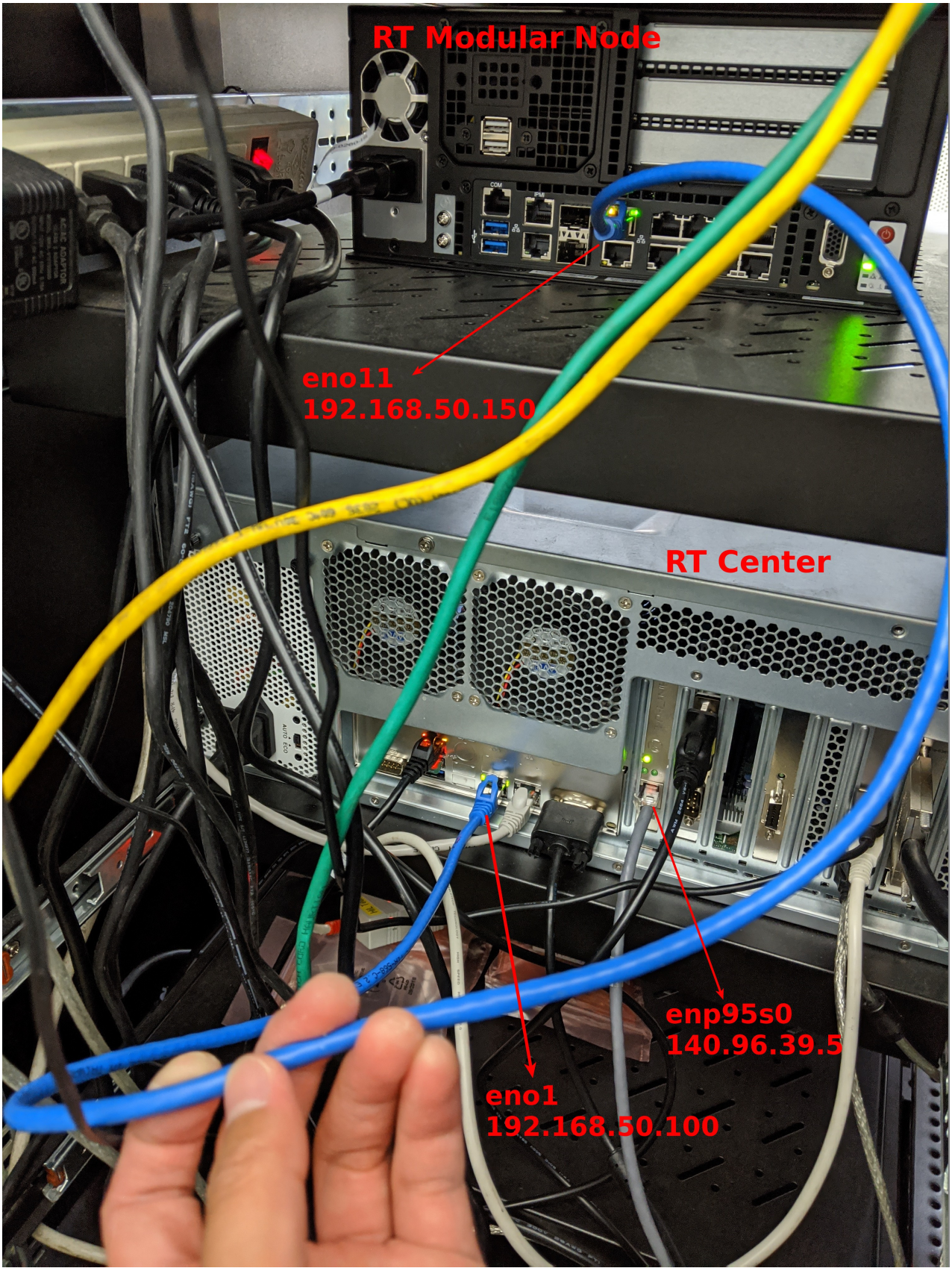
8.1 **LAN Network Setup**

RT Center

- Ethernet interface "eno1": 192.168.50.100
- Ethernet interface "enp95s0": 140.96.39.5

RT Modular Node

- Ethernet interface "eno11": 192.168.50.150


RT Modular Node
eno11
192.168.50.150
RT Center
enp95s0
140.96.39.5
eno1
192.168.50.100

RT Modular Node's eno11 interface (Top) connected to RT Center's eno1 interface (Bottom) through a blue Ethernet.

8.2 Sharing Internet Connection

The RT Center has internet and ideally would be good to share connection to all the RT Modular Nodes that are locally connected. To enable this. forwarding and routing needs to be configured on the RT Center, and the name server has to be set on the RT Modular Nodes. In network's terms, the RT Center would act as a gateway, while the RT Modular Nodes being clients to access the internet via the gateway. In the current setup, the gateway would connect to a client with interface having a static ip of 192.168.50.100, and have another interface, with static ip of 140.96.39.5, connected to the internet. The client would connect to 192.168.50.100 with an interface assigned with static ip of 192.168.50.150.

8.2.1 RT Center Setup (Gateway)

1. Assign static IP to interface eno1
    a.
    ```
    sudo vim /etc/network/interfaces
    ```
    b.
    ```
    auto eno1
    iface eno1 inet static
    address 192.168.50.100
    netmask 255.2555.255.0
    ```
2. Configure iptables for NAT translation
    a. set the forwarding and NAT translation rules via iptables
        i.
        ```
        sudo iptables -A FORWARD -o enp95s0 -i eno1 -s 192.168.50.0/24 -m conntrack --ctstate
        NEW -j ACCEPT
        sudo iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
        sudo iptables -t nat -F POSTROUTING
        sudo iptables -t nat -A POSTROUTING -o enp95s0 -j MASQUERADE
        ```
    b. saving iptables rules to apply at startup
        i.
        ```
        sudo iptables-save | sudo tee /etc/iptables.sav
        ```
    c. Add the following line in /etc/rc.local before the "exit 0" line
        i.
        ```
        iptables-restore < /etc/iptables.sav
        ```
    d. enable routing
        i. enable IP forwarding
        ```
        sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
        ```

ii. add the following lines in /etc/sysctl.conf

```
net.ipv4.conf.default.forwarding=1
net.ipv4.conf.all.forwarding=1
net.ipv4.ip_forward=1
```

3. Check name server
   a. The clients would need to set the same name server as that of the gateway. Use nmcli to search for the name server particularly used by the interface connected to the internet (eno1)

      i.
      ```
      nmcli dev show enp95s0 | grep -i dns
      ```

      ```
      d400@u16-01-RTComputer:~$ nmcli dev show enp95s0 | grep -i dns
      IP4.DNS[1]:                             140.96.254.102
      IP4.DNS[2]:                             140.96.254.96
      d400@u16-01-RTComputer:~$
      ```

8.2.2 RT Modular Nodes Setup (Clients)

1. Assign static IP to interface eno11 through adding lines in /etc/network/interfaces

   a.
   ```
   auto eno11
   iface eno11 inet static
   address 192.168.50.150
   netmask 255.255.255.0
   gateway 192.168.50.100
   ```

      i. here the ip address of gateway is the interface we are connecting to
2. Add route
   a. sudo ip route add default via 192.168.50.100
   b. sudo /etc/init.d/networking restart
3. Set correct dynamic name server (DNS)
   a. add the name server we obtained on the gateway to file /etc/resolv.conf

      ```
      #nameserver 127.0.0.53
      nameserver 140.96.254.102
      ```

   b. restart the network with sudo /etc/init.d/networking restart

References:

1. https://help.ubuntu.com/community/Internet/ConnectionSharing

9 RT HIL Simulation Installation

9.1 Overview

The rt-hil-simulation[129] repository should install the following:

1. RT modules
    a. motor control, or other plain models
    b. Simulink generated code
2. Non-RT modules
    a. dds code
    b. IO interfaces
    c. NodeJS
3. Scripts
    a. Python scripts
    b. Shell scripts

9.2 Compile

rt-hil-simulation can choose what modules to install with CMake. For details of the options check the "option()" commands in the CMakeLists.txt[130] file.

1. Make a build directory where the CMakeLists.txt is at

```
mkdir build
cd build
```

2. Specify the modules to be installed with CMake. If we want to install NI, Pickering, Xenomai, and not install pcan, dds, then we will run the following commands

```
cmake .. -DNI=ON -DPICKERING=ON -DXENOMAI=ON -DPCAN=OFF -DDDS=OFF
```

    a. make sure that the modules to be compiled has its libraries installed. Otherwise CMake will generate errors.
3. Once commands run successfully, run make with the generated makefiles.

```
make
```

9.3 Install

With successful compilations, the compiled modules can be installed:

1. Run the following command to install compiled modules

129 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse
130 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/rt-hil-simulation/browse/CMakeLists.txt

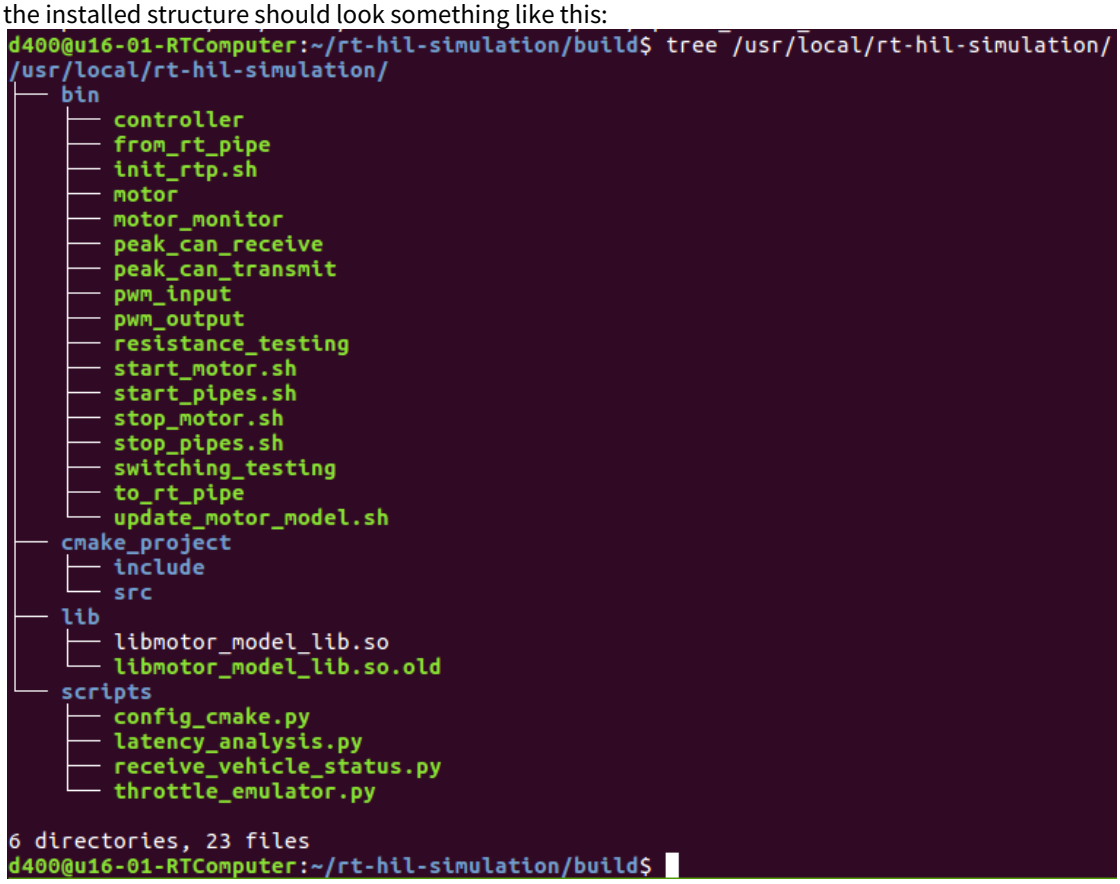
```
sudo make install
```

   a. The installed path can be seen from the "install()" command in CMakeLists.txt with DESTINATION appended with ${install_dir}.

2. One can check for successful installation by checking if "rt-hil-simulation" has been installed under "/usr/local"

```
ls /usr/local
```

   a. the installed structure should look something like this:



9.4 Running

The sequence of running the modules would be the following:

1. Start modules that runs the model (e.g. motor)

```
/usr/local/rt-hil-simulation/bin/start_motor.sh
```

2. Start pipe service that pipes between RT and non-RT (e.g. from_rt_pipe)

```
/usr/local/rt-hil-simulation/bin/start_pipes.sh
```

3. Start services that would interact with the model through the pipes (e.g. throttle_emulator.py)

```
/usr/local/rt-hil-simulation/scripts/throttle_emulator.py
```

If the user would need to stop the processes then

1. Stop/kill the services interacting

```
ctrl + c
```

2. Kill the pipe services

```
/usr/local/rt-hil-simulation/bin/stop_pipes.sh
```

3. Kill the model-related modules

```
/usr/local/rt-hil-simulation/bin/stop_motor.sh
```

9.5 Update Model Shared Library

To Update the motor model do the following:

1. Obtain a CMake project that would compile the new model library
   a. If it's a Simulink model, see From Windows Simulink to Ubuntu CMake from MATLAB Simulink(see page 21) and CMake Project Config and File Collector(see page 27)
2. Once a CMake project with a new model has been obtained, run

```
/usr/local/rt-hil-simulation/bin/update_motor_model.sh [cmake_proj_path]
```

   a. This will compile the CMake proj and installs the newly compiled shared library to the installation of rt-hil-simulation
3. This will update the "libmotor_model_lib.so" under /usr/local/rt-hil-simulation/lib, while saving the previous lib with ".old" appended at the end
4. Once Updated, restart everything so that the new model shared library will be loaded.

10 IO Interfaces

10.1 Installing National Instrument NI PCIe-6361 Driver

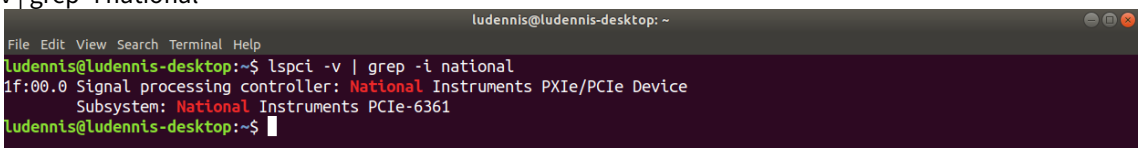
OS: Ubuntu 18.04

Linux Kernel Version: 4.4.182

Xenomai version: 3.0.9

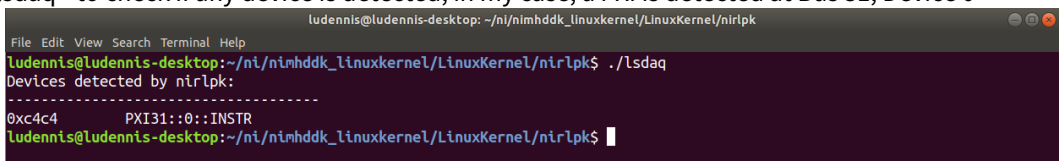
Procedures:

Checking if Ubuntu has detected the PCI card

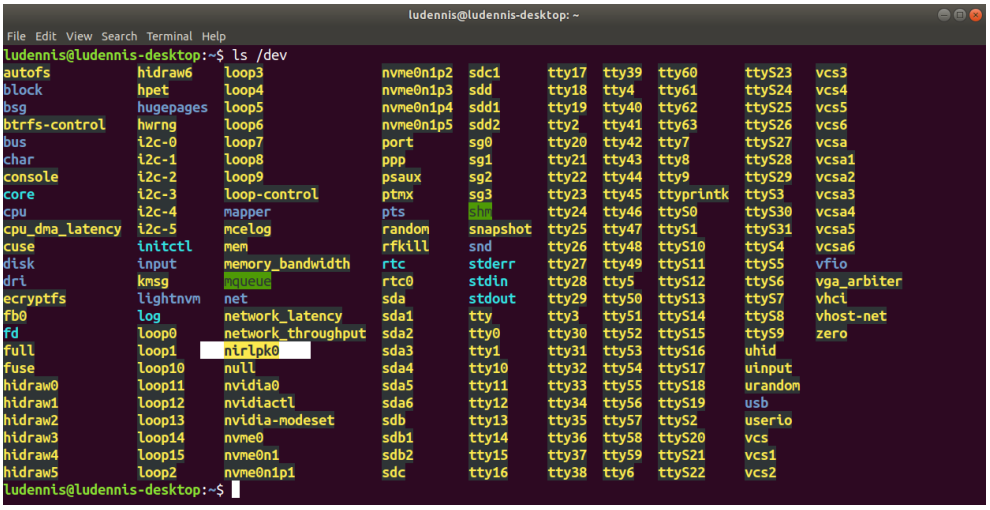
1. `lspci -v | grep -i national`
    a. 

There are two drivers to download from, one driver for the Linux Kernel Module, and the other driver for running cpp code on the card:

Installing Linux Kernel Module Driver for Linux to Use the PCIe-6361

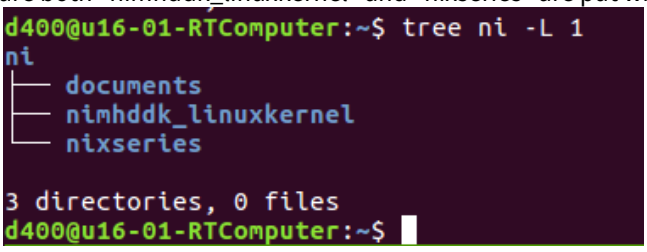
1. git clone driver from http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/nimhddk_linuxkernel/browse
2. checkout branch "linux-kernel-4.4.182"
    a. `git checkout linux-kernel-4.4.182"
3. Enter directory "nimhddk_linuxkernel/LinuxKernel/nirlpk"
4. `make`
5. `make install`
6. After a successful compile and install, check if the kernel module has been successfully installed
    a. `lsmod | grep -i nirlpk`
    b. if there's nothing, that means the kernel module didn't install successfully
        i. check the last 10-15 lines of `dmesg` and debug
7. Check if there's a DAQ (Data AcQuisition) device that can be found by the kernel module
    a. run `./lsdaq` to check if any device is detected, in my case, a PXI is detected at Bus 31, Device 0
        i. 

        ii. here the number "31" in "PXI31" is the bus number
        iii. and the "0" in "::0::INSTR" is the device number
8. Lastly, check if the device is listed in the "/dev" directory
    a. `ls /dev` and search for "nirlpk#"

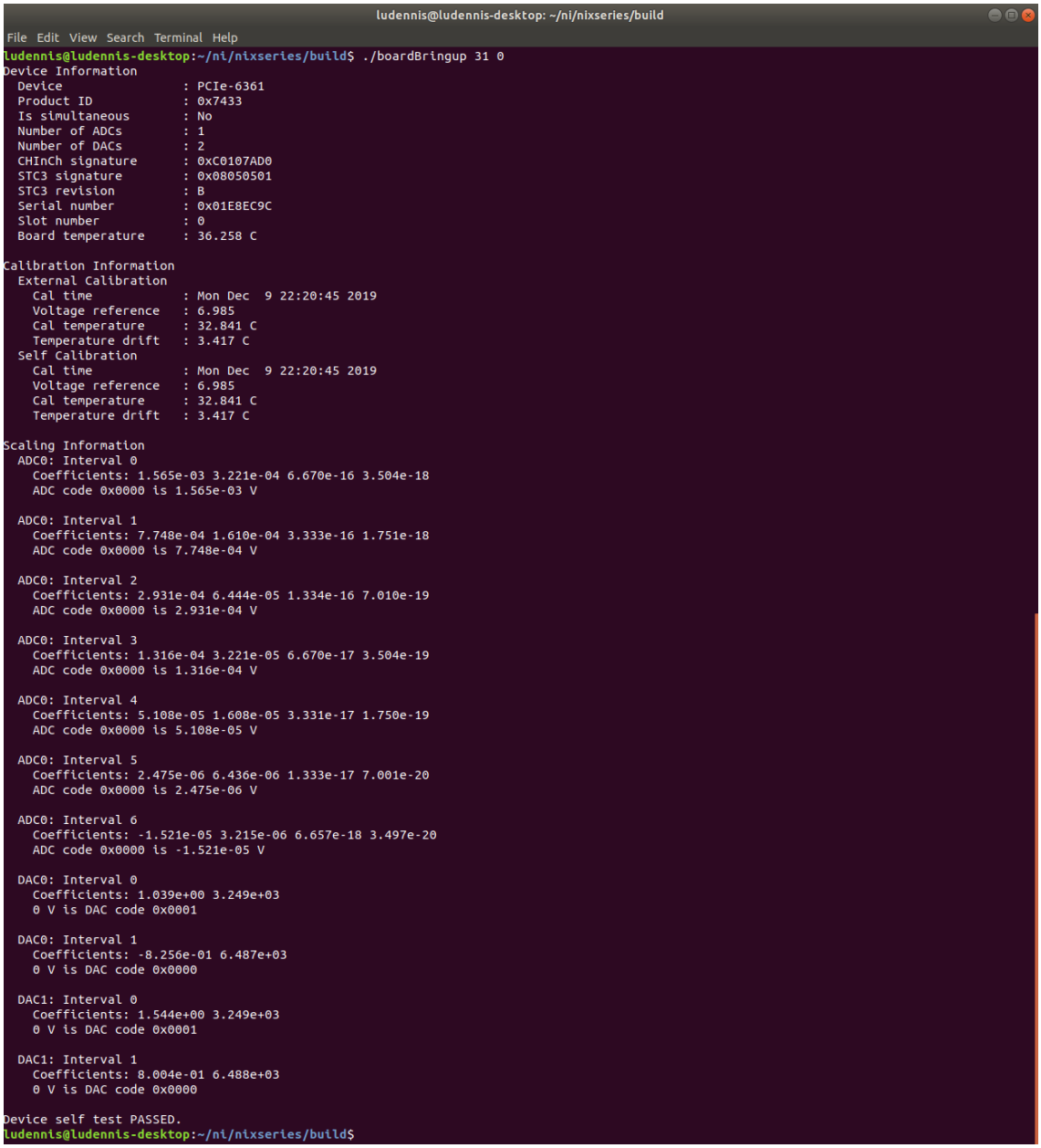
i. 

b. If it's not listed, then go to directory "nimhddk_linuxkernel/LinuxKernel/nirlpk/"
   i. `cd nimhddk_linuxkernel/LinuxKernel/nirlpk`
   ii. `./nirlpk`
c. Check again to ensure that "nirlpk#" is in the "/dev" directory

Installing NI X Series Driver to Operate the PCIe-6361

1. git clone driver from http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/nixseries/browse
   a. make sure both "nimhddk_linuxkernel" and "nixseries" are put within the same directory:
      i. 

2. Enter directory "nixseries"
3. Make a "build" directory
   a. `mkdir build`
4. `make`
5. `cd build`
6. Test if the driver can find and talk to the PCIe-6361
   a. `./boardBringup [ bus # ] [ device # ]`
   b. where bus# and device# are obtained when running `./lsdaq`
   c. the hexadecimal number of bus# and device# can also be found from `lspci`
      i. see https://diego.assencio.com/?index=649b7a71b35fc7ad41e03b6d0e825f07 for how to read the output of `lspci`



d.

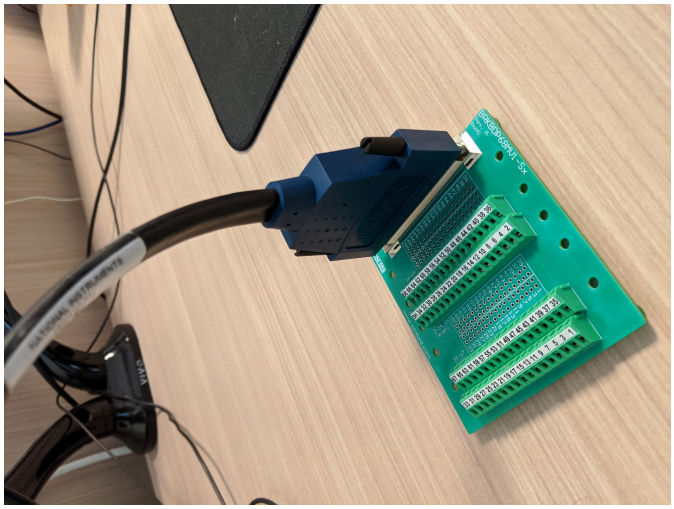
e. an error of "invalid device descriptor will occur when a wrong bus/device number is fed or the "nirlpk#" doesn't exist within directory "/dev"

7. Since nimhddk was written up to Linux Kernel version 2.6, the Linux Kernel we are using (4.4.182) has since changed a little.
   a. Source code modification of file "LinuxKernel/nirlpk/nirlpk.c" can be found in this commit 4a3b6f1ae49[131]

Testing On Breakout Board

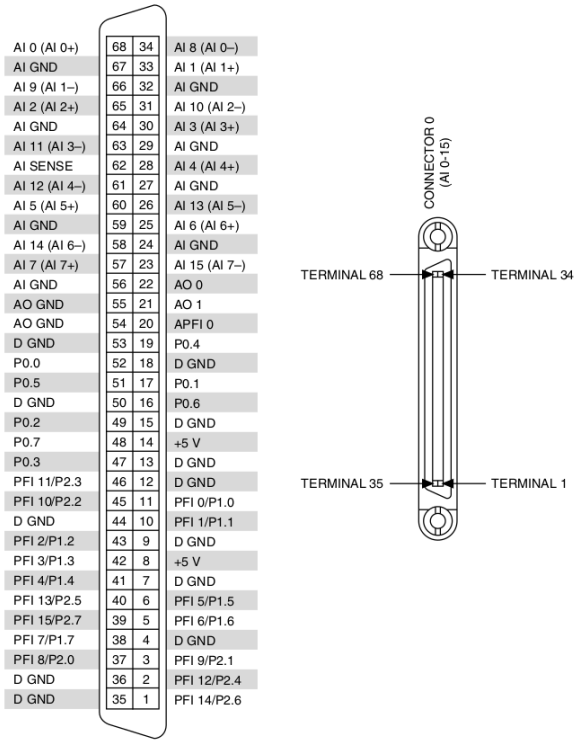
1. connect the connector to a breakout board

131 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/nimhddk_linuxkernel/commits/db3deb69023dfc08a0bc1fb238e4674531179310

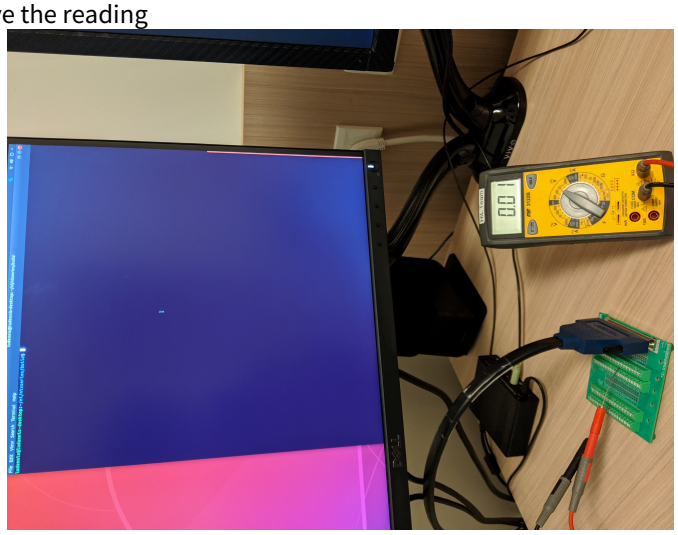
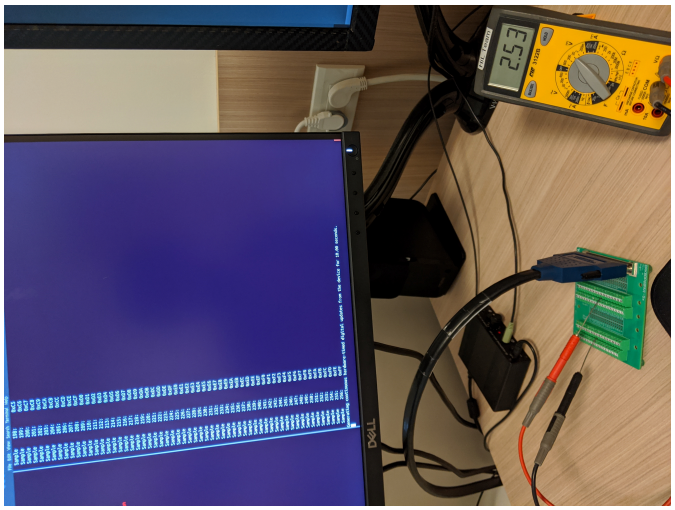
a. 

2. enter "build" directory
3. connect the positive end of the multimeter to the measuring pin, and the negative end to any D GND, following its Pinout from the user manual:

Figure 9. NI PCIe/PXIe-6361 Pinout

a. 

b. The terminals can be categorized as follows
   i. Analog Input (AI 0 - 15)
   ii. Analog Input Ground (AI GND)
   iii. Analog Output (AO 0 - 1)
   iv. Analog Output Ground (AO GND)
   v. Analog Programmable Function Interface (APFI)
   vi. Digital Input Output (P0.0 - 0.7)
   vii. Digital Programmable Function Interface (PFI0/P1.0 - PFI15/P2.7)
   viii. Digital Ground (D GND)

4. `./dioex5` (or `./aoex5` for analog output)
    a. this program will output waveform to all the lines in port0 (P0.0 - P0.7) as digital output
    b. measuring terminals P0.0 - P0.7 with the multimeter should have some voltage when the program is running
5. Observe the reading
    a. 
    
    b. Here pin 52 (P0.0, digital i/o line) is being measured.
6. A table of the result of each pin can be seen here (program was re-runned after each measurement) for DO (digital output) and AO (analog output):
    a. DO (Digital Output):

Port.Line	Pin	GND Pin	Voltage
P0.0	52	53	2.53
P0.1	17	18	2.53
P0.2	49	50	2.52

P0.3	47	50	2.54
P0.4	19	18	0.01
P0.5	51	50	1.99 ~ 3.08
P0.6	16	15	1.83 ~ 3.22
P0.7	48	50	0.01 ~ 5.04

Port.Line	**Pin**	**GND Pin**	**Voltage**
P1.0	11	12	0.01
P1.1	10	9	0.01
P1.2	43	44	0.01
P1.3	42	44	0.01
P1.4	41	44	0.01
P1.5	6	7	0.01
P1.6	5	4	0.01
P1.7	38	36	0.01
Port.Line	**Pin**	**GND Pin**	**Voltage**
P2.0	37	36	0.01
P2.1	3	4	0.01

P2.2	45	44	0.01
P2.3	46	44	0.01
P2.4	2	4	0.01
P2.5	40	36	0.01
P2.6	1	4	0.01
P2.7	39	36	0.01

b. AO (Analog Output):

Port.Line	**Pin**	**GND Pin**	**Voltage**
AO 0	22	55	-0.75 ~ 0.72
AO 1	21	55	-2.47 ~ 2.49

10.2 Installing PEAK CAN PCI FD Card

Install PEAK CAN driver

1. Download repository peak-linux-driver-8.9.0[132]
2. enter into directory "peak-linux-driver-8.9.0"
3. compile the driver with the following make commands
    a. `make all`
    b. `sudo make install`
        i. this will also install a kernel module named "pcan.ko"

Ensure Kernel Modules are in the Current Kernel

1. go to your linux source tree
2. `make menuconfig`
3. ensure the following kernel options is set to either modular (=m) or included (=y)
    a. "PEAK PCAN-PCI/PCIe/miniPCI Cards"
        i. Networking Support → CAN bus subsystem support → CAN Device Drivers → Platform CAN drivers with Netlink support → Philips/NXP SJA1000 devices
    b. "PEAK PCAN-PC Cards"
        i. Networking Support → CAN bus subsystem support → CAN Device Drivers → Platform CAN drivers with Netlink support → Philips/NXP SJA1000 devices

132 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/peak-linux-driver-8.9.0/browse

   c. "PEAK PCAN-USB/USB Pro interfaces for CAN 2.0b/CAN-FD"
      i. Networking support → CAN bus subsystem support → CAN Device Drivers → Platform CAN drivers with Netlink support → CAN USB interfaces
   d. "PEAK PCI Card"
      i. Xenomai/cobalt → Drivers → CAN drivers → RT-Socket-CAN, CAN raw socket interface → Philips SJA1000 CAN controller
4. recompile the kernel and reboot into the kernel, see step 4 in Patching, Building, and Booting into a Xenomai-Compatible Linux Kernel(see page 67)

10.3 Pickering PXI Interface Card

- Accessing Pickering PXI Interface Card with C++(see page 45)
- Accessing Pickering PXI Interface Card with Python(see page 93)

10.3.1 Accessing Pickering PXI Interface Card with Python

1. Download repository pickering-linux-driver[133]
2. enter into directory "pickering-linux-driver/python_api"
3. run `python3 list_pxi_cards.py`
4. for a non-precision resistance card
   a. `python3 nonprecision_resistor.py`
5. for a switching card
   a. `python3 switching.py`
6. to access other kinds of PXI card, one can inherit "class PXICard" from python_api/PXICard.py[134]

133 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/pickering-linux-driver/browse
134 http://mmslswdev.itri.org.tw/bitbucket/users/a70572/repos/pickering-linux-driver/browse/python_api/PXICard.py