QuantiPhi for RL78 and MICON Racing

RL78

Description: Using cutting-edge model-based design tools, you will design a strategy for a Renesas MICON car, a miniature, autonomous electric vehicle. You will take a basic control strategy to the next level with new model-based strategies and calibrations. These strategies will be auto-coded, flashed onto a MICON car, and evaluated on a real track. Plenty of time will be allotted for individual experimentation, questions, and of course, racing.

Lab Objectives
1. Experiment with RL78 and QuantiPhi
2. Learn how MBD can be applied to solve complex problems

Lab Materials
Please verify you have the following materials at your lab station.
- Lab Procedure
- Links on the Desktop to MATLAB, QuantiPhi, UniPhi

Skill Level
1. Understanding of C code
2. Understanding of MATLAB/Simulink

Time to Complete Lab
120 Minutes

Lab Sections
1. Low-Level Drivers with QuantiPhi .................................................................2
2. Micon Race Simulation ..................................................................................15
1 Low-Level Drivers with QuantiPhi

Overview:
In this activity, your goal is to work with a MCU that you’ve (ideally) never seen before and be successful in generating code for a hardware design based around it.

The MCU is an RL78 1014LJ and the hardware design you’ll be targeting is a Micon Rally Car.

At the end of this activity, you will have used QuantiPhi to set up the I/O for driving the Micon steering servo, and an accept a push button input.

Procedural Steps

Step 1.1 From the Desktop, launch QuantiPhi Renesas Edition. The QuantiPhi user interface will then open.
Step 1.2 Your first step is to create a new QuantiPhi project. From the File menu, choose “New Project” and then choose the RL78 project type from the wizard.

Step 1.3 Supply the name for your new project. Use “Micon Lab” for simplicity.
Step 1.4 You should see a “project tree” on the left side of the screen. This gives you a breakout of all of the different subsystems of the RL78 and some useful abstractions.

Your first order of business is to choose “Microcontroller Variant.” Double-click it in the project tree!
Step 1.5 You can consult the schematic located on your desktop to find the desired variant the hard way or you can look in the info box:

The variant used on this Micon Rally Car is a G14 R5F104LJ. It's a 256K part with 64 pins.

<table>
<thead>
<tr>
<th>Part number</th>
<th>Family</th>
<th>pins</th>
<th>Program ...</th>
<th>RAM (by ...</th>
<th>ADC</th>
</tr>
</thead>
<tbody>
<tr>
<td>R5F104JF</td>
<td>G14</td>
<td>52 pins</td>
<td>96 kb</td>
<td>12288 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F1043G</td>
<td>G14</td>
<td>52 pins</td>
<td>128 kb</td>
<td>16384 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F1043H</td>
<td>G14</td>
<td>52 pins</td>
<td>192 kb</td>
<td>20480 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104JJ</td>
<td>G14</td>
<td>52 pins</td>
<td>256 kb</td>
<td>24576 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LC</td>
<td>G14</td>
<td>64 pins</td>
<td>32 kb</td>
<td>4096 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LD</td>
<td>G14</td>
<td>64 pins</td>
<td>48 kb</td>
<td>5632 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LE</td>
<td>G14</td>
<td>64 pins</td>
<td>64 kb</td>
<td>5632 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LF</td>
<td>G14</td>
<td>64 pins</td>
<td>96 kb</td>
<td>12288 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LG</td>
<td>G14</td>
<td>64 pins</td>
<td>128 kb</td>
<td>16384 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LH</td>
<td>G14</td>
<td>64 pins</td>
<td>192 kb</td>
<td>20480 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104LJ</td>
<td>G14</td>
<td>64 pins</td>
<td>256 kb</td>
<td>24576 b</td>
<td>12</td>
</tr>
<tr>
<td>R5F104MF</td>
<td>G14</td>
<td>80 pins</td>
<td>96 kb</td>
<td>12288 b</td>
<td>17</td>
</tr>
<tr>
<td>R5F104MG</td>
<td>G14</td>
<td>80 pins</td>
<td>128 kb</td>
<td>16384 b</td>
<td>17</td>
</tr>
<tr>
<td>R5F104MH</td>
<td>G14</td>
<td>80 pins</td>
<td>192 kb</td>
<td>20480 b</td>
<td>17</td>
</tr>
<tr>
<td>R5F104MJ</td>
<td>G14</td>
<td>80 pins</td>
<td>256 kb</td>
<td>24576 b</td>
<td>17</td>
</tr>
<tr>
<td>R5F104LF</td>
<td>G14</td>
<td>100 pins</td>
<td>256 kb</td>
<td>12288 b</td>
<td>20</td>
</tr>
</tbody>
</table>

Select the variant. Your selection will automatically take effect.

Step 1.6
Your next step will be to configure a PWM output that will be used to drive the steering servo. Consult the Lab Companion documents that describe the interface between the PCB with the RL78 and the various motors and sensors on the vehicle.

Locate the pin/PWM channel that is connected to the SERVO.

**Finding this information “the hard way” more closely approximates a real world scenario.**

**Step 1.7** Locate the PWM channel in the QuantiPhi, PWM configuration window

**Step 1.8** This servo requires a 62.5Hz control signal, so configure the servo steering PWM channel for 62.5Hz. At least 12 bits of resolution should give us enough for -90 to +90 degrees that the servo is capable of with at least a 1 degree resolution.

**In actuality, QuantiPhi will be able to set up your clock tree to achieve a much better resolution. QuantiPhi uses “yellow panels” to give you feedback about how it was able to meet or exceed your expectations.**
Step 1.9  Let QuantiPhi lead you to a successful configuration with its “Real-Time Validation” system. Use the “Show Me” button that accompanies the validation messages to take you right to the correct place in the UI where changes can be made.

Step 1.10  The next step will be to make sure that we label the input P17 as our Push Button input. Locate the Pin Assignments tab and double click on it.

Step 1.11  Locate P17 (we’ll save you the time of looking up that this is the push button switch on the Micon Rally car) using the search bar. Assign it a name of “PUSH BUTTON”.

Labeling it as PUSH_BUTTON will mean that when code is generated, a convenient macro will be generated that uses the name.
Step 1.12 We’re going to need a time base for our test application. Let’s say a tick of 2ms will be sufficient. For that, we’ll need a General Purpose Timer (GPT). Locate the GPT area of the product and open it up.

Step 1.13 Configure gpt0 with a name of “Scheduler Tick” and tell QuantiPhi that you’re going to need it to figure out how to achieve a 2ms timeout. Use the real-time validation system to guide you.
Let's say you're curious about interrupts that are being used and the priorities. Visit the "Interrupts" window in your project. Those interrupts with the QuantiPhi icon in the "Used" column that are fully opaque are those that the generated code will be using. You can adjust the priority here.

```c
#pragma interrupt ISR_INTTM00
void ISR_INTTM00(void)
{
   /* User-specified Pre-ISR Hook */

   C language function name

   /* QuantiPhi ISR Handling */

   /* User-specified Post-ISR Hook */

   C language function name
```

- **GPT Channel gpt0 - Scheduler Tick Internal ISR**
  - Type: Driver Internal
  - Active: Yes

- **GPT Channel gpt0 - Scheduler Tick Driver Callback**
  - Type: Driver Callback
  - Active: Yes
Let’s say you’re also curious about how the RL78 clock tree was configured to meet your PWM and GPT requirements so far. Visit the Advanced Timer Configuration window to find out!
Step 1.14 Now it’s time to generate our low-level code. Visit the Code Generation window to adjust where the code is generated. This area of the product tweaks the code generation process.

Simply note the code generation directory of “C:\QuantiPhi”. This will be sufficient for this lab.

Step 1.15 Finally, click the “Generate Code” button on your toolbar (or press CTRL-B). An output window will appear to update you on the generation progress.
Step 1.16 Now that you have driver code, we’re going to integrate it into a test application that should 
exercise the Micon car’s steering system. Locate and double-click the link to 
“Activity1_CubeSuiteProject.mtpj” on your desktop.

Step 1.17 There is already a placeholder folder and main.c file in this project. It just needs the files you 
just generated to compile and link. Right-click on this folder and choose Add > Add File… 
Add all .c and .h files from C:\quantiph

Step 1.18
Take a look at some of the generated drivers and at main.c. See how you translated our system requirements into a QuantiPhi configuration which can be accessed in code. Example:

```c
/* Set up gpt0 to give us a callback every 2ms
 * Instead of using gpt0, we use QP_GPT_CHANNEL_SCHEDULER_TICK, which was a define
 * generated for us from the name you supplied.
 */
GPT_SetTimerCallback(QP_GPT_CHANNEL_SCHEDULER_TICK, (qp_gpt_callback_t *)gpt_callback);
GPT_ConvertHertzToTicks(QP_GPT_CHANNEL_SCHEDULER_TICK, 500U, &gpt_ticks);
GPT_StartTimer(QP_GPT_CHANNEL_SCHEDULER_TICK, (uint16_t)gpt_ticks, 0U);
```

Step 1.19 Time to build your project! Either the button on the toolbar or the F7 key will work.

![Builds the project. (F7)](image)

Step 1.20 Resolve any issues with the compilation. We know that everything always goes perfectly and you shouldn’t have any issues... but in case you do, here are some common problems:

1. The compiler can’t make sense of QP_GPT_CHANNEL_SCHEDULER_TICK. This means that the name you gave the GPT channel didn’t quite match what we expected. Make sure that it’s named “Scheduler Tick” (case doesn’t matter).

2. There are assorted missing symbols starting with qp_. Did you make sure to add _all_ of the files (both .c and .h) from the code generation directory? You can always go back and make sure.
STOP

The next section of the lab involves fun and excitement. There are important things to learn, so please wait for further instruction before proceeding.

Just let the instructor know that you are ready with a shout, turn your die to “2”, or both.
2 Micon Race Simulation

Overview:
In this second activity, you will be learning what you need to know in order to create a control strategy that successfully leads the Micon Rally Car around a track.

SimuQuest has developed a simulation environment within MATLAB/Simulink that allows you to quickly prototype your algorithms and evaluate them in simulation. You will learn how to use this environment as well as generate code.

Procedural Steps

Step 2.1 Start MATLAB by double-clicking the “MATLAB R2010b” icon on the desktop. The splash screen should appear followed by the main interface.

Make sure that you are in the C:\SimuQuestDevConLab\work directory and that the following text appeared in the Command Window (version numbers may change):

```
Starting UniPhi 5.0.5.5772+ [Beta]
------------------------------------------------------------
```

```
Setting the path... done!
Initializing core... done!
Restoring UniPhi project paths... done!
Bringing up MATLAB Interface... done!
Custom Extensions...
------------------------------------------------------------
```

```
версия продукта: QuantiPhi: Renesas Edition
версия продукта: 1.3.0.6141
```

```
Setting the path... done!
Testing installation... done!
Initializing... done!
Checking Version... 1.3.0.6141
Release Status... Engineering Release
Locating supported compilers... done!
Restoring project paths... done!
```

By default, the MATLAB installation should automatically take you to this directory and run some initialization commands. If not, please contact the instructor.

Step 2.2 Your first goal is to simulate the model you’ve been given. Choose the “one_corner” track by running the following command in the MATLAB command window:

```
>> qp_micon_track use track one_corner
### Track loaded: one_corner
```
Step 2.3  

Next, open the model, **micon_simulation** by double-clicking it in the Current Directory window or by typing it on the command line.
Step 2.4 Simulate the model by pressing the “play” button.

A custom Micon Racing simulation UI will appear shortly:

Step 2.5 Press the “Open Start Gate” button underneath the simulated LEDs. The car should begin to move. You will notice the speed updating and the LEDs and Raw: fields showing you what your algorithm sees.
These values update during simulation. You can add anything you want to see to this table easily using the example provided in the MICON Controls subsystem (later!).

**Step 2.6** So! The car flew off the track. But it did it so slowly! Let’s speed things up a bit and in the process, learn about our data dictionary.

Double-click here:

![Diagram](image)

**Step 2.7** Take notice of the comments in this subsystem. We’ve implemented some big pieces that you’ll need to be successful already (we do only have 2 hours after all). When you’re done looking around, open the “Main Tracking Controls” subsystem:
Step 2.8  See how it’s taking in a RaceOk signal (don’t mess with this!), a Position Code between -40 and +40, an unconditioned version of same, and a bunch of “cross-line signals”. We went over these earlier.

You’ll now notice the reason we just went straight ahead at a slow power. No one changed the steering angle from 0 degrees and we have a constant power. Let’s see what that value is.

We'll of course this won't work! I'll drive right off the track. It does, however, give you an example of how you can use calls and signals centrally in UniPhi.
Step 2.9  Double click K_EXAMPLE_POWER:

See why we went so slowly? We were at 50% power. Let’s change it to 80.

Step 2.10  Click on the edit button (the one with the pencil and the arrow):

That should open UniPhi (our data dictionary for this project) and open the Parameter.
Step 2.11 Change the Initial Value to 80, hit enter, and save the Parameter (CTRL-S).

Step 2.12 Run your simulation again. This time, you can use the play button in the UI.

Was it faster?
Step 2.13 Faster, but still driving off the track. Let’s fix that. Go to the top level of the model and double click on the “Click Here for Useful Blocks” button.

Step 2.14 Copy (CTRL-C or use menu) the Sample Controls block and paste it into your controls subsystem like so:

Step 2.15 Simulate your system again. See how it actually tracks? This is because we’ve produced a position code already between -40 and +40 that gives you roughly the correct steering angle to command in the correct direction to compensate.

Controls engineers will recognize this as simple proportional control.
Step 2.16  Do the same with the other setup in that collection of blocks. It gives you an example of how to achieve a less-than-one gain (will not wobble so crazily) by using Simulink's fixed point support.

These blocks are set up to store the values as 100x internally. So 0.5 becomes 50. All the math works out and it will run on the target (can’t say that if we tried to emulate floating point in software!)

Step 2.17  Now that we have an albeit “wobbly” strategy, the next step is to use QuantiPhi to generate code for it that is flashable onto the car. Open the model micon_code using either the command window or the Current Directory window.
Everything is set up in this model for code generation. It just needs your control strategy. Normally, the process of taking features to code does not require you to copy in your controls, but this is the simplest approach for the lab.

Step 2.18 Let’s see what’s different about this QuantiPhi project. Double-click the QuantiPhi Connector block at the top level and then click on Open QuantiPhi.

Step 2.19 First notice the tool had a variant already that specially targeted the Micon car.
Step 2.20  Next, visit the Micon configuration window by double clicking on “Micon” in the project tree. QuantPhIs shows you the I/O of the vehicle and offers opportunities to customize it for non-stock configurations.

Step 2.21 Code generated from this project would generate a set of high-level drivers for Micon that we’ll use in our model (or in hand code). Go back to the code model and find the “Inputs” subsystem. Within it is an example of blocks that access the Micon high-level driver API.
Step 2.22 At the top level of the micon_code model, double-click the “Build Model” button.

The build procedure is automated:

```
Command Window
Starting...
____________________________________________
QuantiPhi Code Generator
Version: 1.3.0.6141
____________________________________________

Code successfully generated into C:\work\Dev\eka2011\trunk\lex\micon\micon_code
... QuantiPhiRE code generation completed.
.
### Caching model source code
..........................................................

..............
### Writing header file micon_code_types.h
### Writing header file micon_code.h

.
### Writing header file micon_code_private.h
### Writing source file micon_code.c
### Writing source file micon_code_data.c

.
### Writing source file main.c
### TLC code generation complete.
### generate TLC report done.

...
```

```
### Successful build of micon_code
[OK] Code Generated
[OK] Compiled using the "Renesas CA78KOR" compiler
____________________________________________
```

Step 2.23  Take this opportunity to browse through the code that was generated in the HTML viewer:
STOP

You are now free to experiment with the simulation strategy. Extra time is available to try one’s strategy out on the real vehicle!

Please turn your die to “3”.

Other tracks are available for simulation! The “lab” track is a tad more interesting. You can post your race times for this track. Will you be the fastest?

>> qp_micon_track use track lab