RF SIGNAL SWITCHING SYSTEM FOR ELECTRON BEAM POSITION MONITOR UTILIZING ARM MICROCONTROLLER

Tomonori Toyoda, Kenji Hayashi, and Masahiro Katoh, IMS, Okazaki, Japan

Abstract

In 2012, for commissioning of UVSOR-III [1], 750-MeV synchrotron radiation facility of the Institute for Molecular Science, we developed a radiofrequency (RF) signal switching system. The main specifications are as follows: (1) it is a very simple and low-cost system; and (2) it can be controlled through LAN. To achieve these goals in limited development time, we adopted "mbed," a 32-bit ARM microcontroller development kit. Using "mbed" as an HTTP server, which stores the HTML (and CSS) files that are included a JavaScript library "mbedRPC.js," we can control this system from a Web browser. Using this system, we efficiently obtained the required parameters in the commissioning.

In the presentation, we report the overview of the RF signal switching system using "mbed" and the method to build the application of LAN control using "mbedRPC.js."

DESIGN OF RF SIGNAL SWITCHING SYSTEM

At the commissioning, particularly before the success of the beam storage, it is important to determine the number of turns that the beam has circulated, where the beam has lost. It is also important to measure the orbit parameters before the storage, such as closed orbit or betatron tunes. Most of these can be realized by beam position detection by using BPM. On the other hand, such a system is not necessary in the daily operation. Therefore, we decided to construct an RF signal switching system as follows:

- We designed it to be as low-cost and simple as possible.
- We used an existing digital oscilloscope for waveform observation and recording.
- We used existing RF cables that are normally used for connecting the BPM heads to the Bergoz signal processing system.
- We set the oscilloscope in the ring and controlled it through LAN.
- We developed a signal switching box that can be controlled through LAN.

The RF signal switching box that we developed is shown in Fig. 1.

OVERVIEW OF RF SIGNAL SWITCHING SYSTEM

Because the BPM signal is a weak (tens of mV to sub-mV) high-speed pulse (one hundred ps bunch length corresponds to several GHz in the frequency domain) caused by electron beams circulating at approximately 90 MHz, we should prevent signal attenuation by switching the signal as much as possible.

Figure 1: Signal switching box (front view).

We also considered the use of semiconductor switches. However, as shown in Table 1, because the losses associated with coaxial switches were generally less and had wider frequency bands than those of semiconductor switches, we decided to use coaxial switches.

Table 1: RF parameters of coaxial switches [2] and certain semiconductor switch

Products	Frequency	Insertion Loss (max)
CCR-33-506	DC-6GHz	0.2 dB
CCR-38	DC-6GHz	0.2 dB
Semiconductor	DC-2.5GHz	3.8 dB

The total number of BPM heads in UVSOR is 24, but because the development time was limited, we constructed a system in which 8 BPM heads can be treated. The block diagram of the RF signal switching system is shown in Fig. 2.

First, we select $4 \times 4 = 16$ from $8 \times 4 = 32$ BPM signals by 16 SPDT (Single-Pole Double-Throw) type coaxial switches. Next, we select $1 \times 4 = 4$ by 4 SP4T (Single-Pole 4-Throw) type coaxial switches.

We controlled the switching of coaxial switches using "mbed," which is a 32-bit ARM microcontroller development kit produced by NXP semiconductors [3]. 74ACT04 and MOSFETs were connected to the outputs of "mbed" to drive the coaxial switches. In addition, we connected a 4-channel digital oscilloscope (5 GHz

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7

sampling frequency, 1 GHz/2 GHz analog signal band) to a LAN. By controlling the I/O ports of "mbed" and a digital oscilloscope, we recorded the BPM signals to a digital oscilloscope selected by the RF signal switching system and the data was analyzed offline.

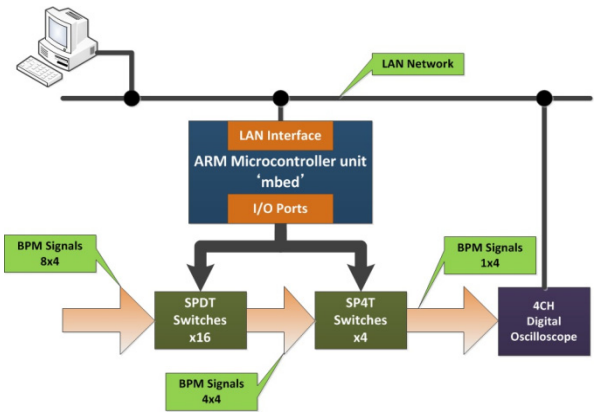

Figure 2: Block diagram of BPM system.

OVERVIEW OF "MBED" PROGRAMMING

We programmed the "mbed" using the C language, but we do not need to be aware of the existence of registers. For example, when setting the digital inputs, we can read the value of 5 pins using the variable "enable," which is defined as "enable = DigitalIn enable(p5)".

With respect to realizing LAN control, if we used it as an HTTP server for example, we can load the libraries (provided free of cost on the official page) and define the network information such as the IP address. Because "mbed" has already been mounted on an Ethernet physical layer IC, we can complete the hardware preparation by connecting it to an RJ-45 modular jack and pins of "mbed."

Instead of not needing to be aware of the existence of the registers, we cannot precisely set operating conditions, and nor can we reduce the consumption current. Therefore, "mbed" is useful for prototyping and verifying operations by constructing the desired functions for applications to quickly construct functions within a limited time such as this equipment.

CONTROLLING "MBED" USING JAVASCRIPT LIBRARY, "MBEDRPC.JS"

The LAN control applications of "mbed," such as TCP/IP, UDP, and HTTP server, are available. On account of the limited development time, we adopted an HTTP server.

When we use an HTTP server to control the "mbed," we store the HTML (and CSS) files that construct the control application to "mbed." In general, we write JavaScript functions using XMLHttpRequest, which is an application programming interfafce (API) of JavaScript.

It is necessary to simultaneously control the coaxial switches in an RF signal switching system, but when using XMLHttpRequest, the outputs of "mbed" became unstable, and we were unable to properly control the coaxial switches. This is because the XMLHttpRequest can be used only once in a JavaScript function to control the outputs of "mbed." Therefore, we adopted a JavaScript library "mbedRPC.js" [4].

To control the I/O ports of "mbed" using "mbedRPC.js," we followed the following procedures.

First, we include "mbedRPC.js" in the HTML file (line 94), as shown in Fig. 3. Also, we define I/O ports and assign them to the variables (lines 95 to 106). Then, it is necessary to write "mbed = new HTTPRPC()" at the beginning of the definition for initialization.

```
94 <script src="mbedRPC.js" language="javascript"></script>
95 <script type="text/javascript">
96         mbed = new HTTPRPC();
97         led2 = new DigitalOut(mbed, LED2);
98         spdt = new DigitalOut(mbed, p8);
99         sp4t1 = new DigitalOut(mbed, p10);
100        sp4t2 = new DigitalOut(mbed, p12);
101        sp4t3 = new DigitalOut(mbed, p14);
102        sp4t4 = new DigitalOut(mbed, p16);
103        function print(str) {
104            document.getElementById("t").innerHTML = str;
105        }
106 </script>
```

Figure 3: Including "mbedPRC.js" and Definitions

Next, we write the operation buttons in HTML (Fig. 4). At this time, we prepare one button to switch ON/OFF the LED on the "mbed" to confirm communication and eight buttons to switch the BPM signals. We use the input tag enclosed in the form tag, such as an e-mail form, and write user actions within the input tags to call JavaScript functions; in this case, the operation of clicking a button using the onclick attribute (line 283 etc.). In Fig. 4, when we click the "LED2 (simple ON/OFF for checking)" button, "button_push," which functions as an argument 0, it is called (line 283), and when we click the "BPM Inputs 1" button, "button_bpm1," which functions as an argument 0, is called (line 288).

```
280 <form name="Form" action="#">
281     <div class="ledcontrol">
282     LED2(simple ON/OFF for checking):
283     <input type="button" value="ON" name="FormButton" onclick="button_push(0)"><br />
284     </div>
285
286     <div class="bpmcontrol">
287     BPM Inputs 1:
288     <input type="button" value="ON" name="FormBPM1" onclick="button_bpm1(0)"><br />
289     BPM Inputs 2:
290     <input type="button" value="ON" name="FormBPM2" onclick="button_bpm1(1)"><br />
291     BPM Inputs 3:
292     <input type="button" value="ON" name="FormBPM3" onclick="button_bpm1(2)"><br />
293     BPM Inputs 4:
294     <input type="button" value="ON" name="FormBPM4" onclick="button_bpm1(3)"><br />
295     BPM Inputs 5:
296     <input type="button" value="ON" name="FormBPM5" onclick="button_bpm1(4)"><br />
297     BPM Inputs 6:
298     <input type="button" value="ON" name="FormBPM6" onclick="button_bpm1(5)"><br />
299     BPM Inputs 7:
300     <input type="button" value="ON" name="FormBPM7" onclick="button_bpm1(6)"><br />
301     BPM Inputs 8:
302     <input type="button" value="ON" name="FormBPM8" onclick="button_bpm1(7)"><br />
303     </div>
304 </form>
```

Figure 4: Descriptions of operation buttons

We have described the operation in the JavaScript function (Fig. 5). "button_bpm1" functions by switching the coaxial switches branched in the switch–case syntax according to arguments 0–7 (otherwise, SP4T type switches OFF to not outputs). For example, in case of argument 0, variable "switchspdt" is set to 0, variables "switchsp4t1–switchsp4t4" are each set, exit from case

ISBN 978-3-95450-139-7

Copyright © 2014 CC-BY-3.0 and by the respective authors

syntax (line 136–140). If necessary, we rewrite the display of the button using the document object model (DOM) of HTML (line141–148). In this case, because group 1 of BPM signals was selected, we switch the display to "OFF" in that "Display will be turned OFF if you click the button again."

```
133  function button_bpm1(flag) {
134      switch (flag) {
135      case 0:
136          switchspdt = 0;
137          switchsp4t1 = 0;
138          switchsp4t2 = 1;
139          switchsp4t3 = 1;
140          switchsp4t4 = 1;
141          document.Form.FormBPM1.value = "OFF";
142          document.Form.FormBPM2.value = "ON";
143          document.Form.FormBPM3.value = "ON";
144          document.Form.FormBPM4.value = "ON";
145          document.Form.FormBPM5.value = "ON";
146          document.Form.FormBPM6.value = "ON";
147          document.Form.FormBPM7.value = "ON";
148          document.Form.FormBPM8.value = "ON";
149          break;
```

Figure 5: Descriptions of controlling

After exiting from the switch–case syntax, we write values that we set in the switch–case syntax for input and output variables that we define in Fig. 3 (Fig. 6). As a result, pin 8 of "mbed" that controls the SPDT-type switches and pin 10 of "mbed" that controls terminal 1 of the SP4T type switches become "L" level; furthermore, pin 12, pin 14, and pin 16 controlling the other terminals of SP4T-type switches become "H" levels. The coaxial switches are switched through 74ACT04 and MOS FETs, group 1 of BPM signals is selected and outputs from SP4T-type switches.

```
271      spdt.write(switchspdt);
272      sp4t1.write(switchsp4t1);
273      sp4t2.write(switchsp4t2);
274      sp4t3.write(switchsp4t3);
275      sp4t4.write(switchsp4t4);
```

Figure 6: Writing input and output variables

PERFORMANCES

An example of the observed BPM signal waveform is shown in Fig. 7. An example of the injection beam trajectory is shown in Fig. 8. The data are simply processed by Eqs. (1) and (2) using the peak voltage obtained from the waveform, as shown in Fig. 7, and no compensation is applied for the nonlinear responses.

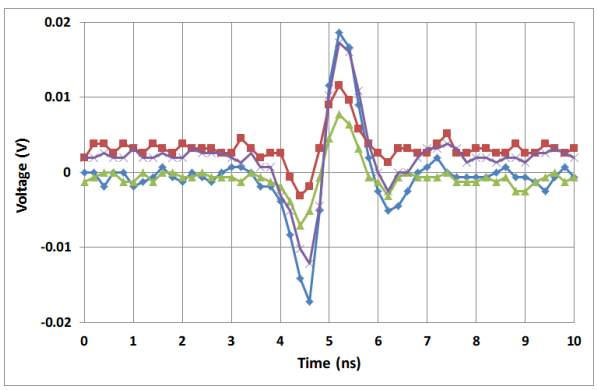

Figure 7: An example of BPM signal waveforms from a BPM head.

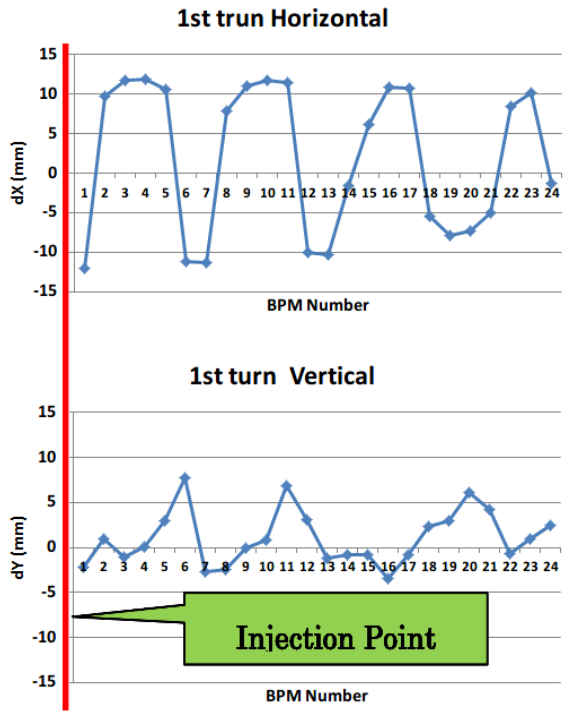

Figure 8: Example of measured injection beam trajectory.

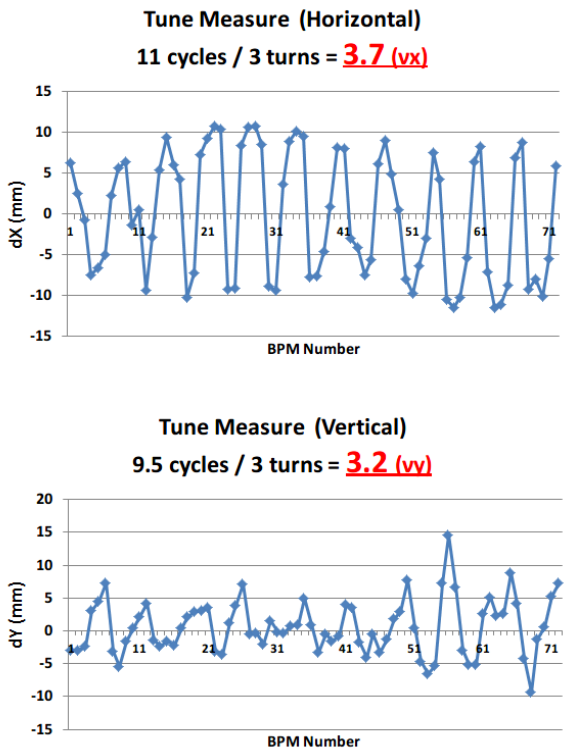

Figure 9: Measured beam position for first three turns after injection.

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7

The betatron tune could also be determined by observing the multiturn orbit. At present, ν_x is 3.7 and ν_y is 3.2 (Fig. 9). This was very useful during the commissioning.

An example of the accuracy check is shown in Fig. 10 and was obtained using a stored beam. The fluctuation is typically in the order of sub millimeters, whereas periodic fluctuations are clearly seen. The period is five turns, and this is simply because of the fact that the beam revolution period is not an integer multiple of the sampling period of oscilloscope. Divergence in the position data should also depend on the mismatch of the lengths of each cable for four electrodes.

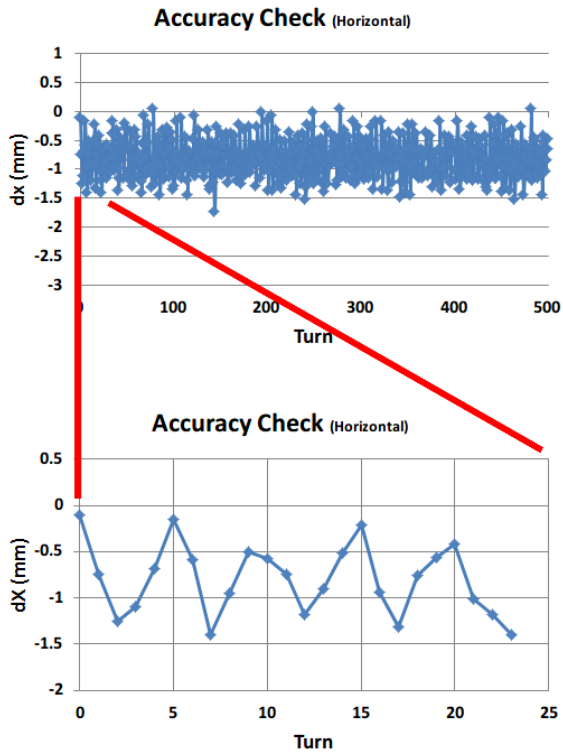

Figure 10: Turn-by-turn BPM data of the stored beam.

CONCLUSION AND FUTURE PLAN

We developed an RF signal switching system over a LAN using coaxial switches, “mbed,” and the JavaScript library “mbedRPC.js.” We demonstrated that the system successfully realized beam storage at the commissioning.

Future works to improve the BPM system are as follows.

First, for example, by performing an analysis using LabVIEW of National Instruments, it is possible to more accurately calculate the peak value and improve the accuracy of the beam position.

Second, we will expand the switching system to deal with 24 BPMs. Another switching system will be used to switch signals to an ordinary BPM or a turn-by-turn BPM.

If we take advantage of the capabilities of the LPC1768 or LPC1769 (the only difference between the two is the maximum operating frequency), we can expand to switch 24 BPM signals. As the first stage, we have developed the base board to facilitate the development of LPC1769 (Fig. 11).

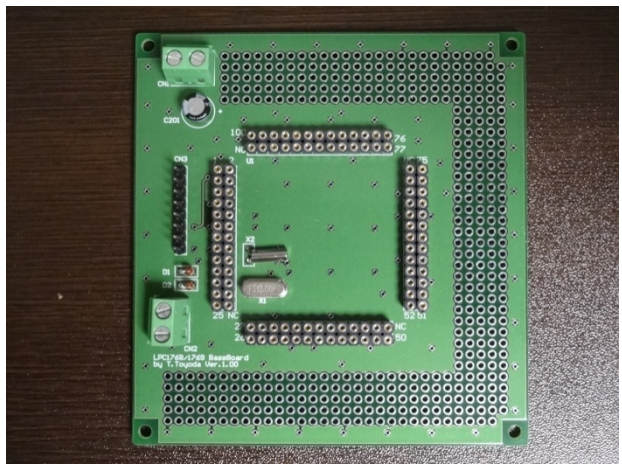

Figure 11: LPC1769 Base board (Board only).

We used only a 12 MHz crystal oscillator for the system clock, a 32.768 kHz crystal oscillator for the real-time clock, a DC 3.3 V power connector, and a programmer Serial Wire Debugger (SWD). We have extensively placed prototyping area as possible to prototype and develop in accordance with peripherals and other circuits.

It is possible to transfer the program by adding a slight modification LPCXpresso [5] evaluation board.

We have so far learnt about GPIO, timers, UART, and A/D conversion using this base board. We would like to learn about how to use TCP/IP by employing the Ethernet physical layer IC and connect it to improve the RF signal switching system.

Third, we developed a fully automatic and fast data acquisition and fully automated data processing.

By combining the analysis of the wave height using LabVIEW and TCP/IP control developed on the LPC1769 base board, we hope to fully automate the data acquisition or analysis and allow more efficient commissioning.

REFERENCE

[1] M. Adachi et al., J. Phys. Conf. Ser. 425(2013) 042013
[2] CCR-33 Series and CCR-38 Datasheets, TELEDYNE COAX SWITCHES; http://www.teledynecoax.com.
[3] “Appendix A High-performance microcontroller! specification of ‘mbed’”, “Introduction of ultra-handy microcontroller ‘mbed’”, p.116, CQ Publishing (in Japanese).
[4] http://mbed.org/cookbook/Interfacing-with-JavaScript.
[5] http://www.lpcware.com/lpcxpresso.

ISBN 978-3-95450-139-7

Copyright © 2014 CC-BY-3.0 and by the respective authors