Device and Driver Support for F3RP61 (Rev. 1.0)

J. Odagiri

Contents

1. Overview

This device and driver support can be used to run EPICS iocCore on an embedded Linux controller, F3RP61 (e-RT3 2.0), made by Yokogawa Electric Corporation. The controller, F3RP61, can access any I/O channel of any I/O module of FA-M3 PLC on the PLC-bus (base unit). This feature opens way for making an FA-M3 PLC itself a new type of IOC. The device and driver support interfaces the iocCore with the I/O modules as well as sequence CPUs that runs on the same base unit. The device and driver support is implemented by wrapping the APIs of the kernel-level drivers, which are included in the Board Support Package (BSP) of F3RP61.

The device and driver support offers only primitive method to access relays and registers of the I/O modules and sequence CPUs. In the sense that any relay and any register can be accessed by the device and driver support, it is universal. However, in order to handle a special module that requires some sequence logic to execute the I/O operation, the sequence logic needs to be implemented by using an EPICS sequencer program by the user. (The sequence logic to handle a special module is implemented by using a ladder program when a sequence CPU is used to execute the I/O operation. The

EPICS sequencer program is used to replace the ladder program.) If some initialization is required on a special module, it can be done by using an EPICS sequencer program, or a runtime database comprised of records that have PINI field value of “YES”.

An F3RP61 can work as an IOC with/without sequence CPUs that run ladder programs. If no sequence CPU is on the base unit, the F3RP61 should manage all I/O activities. If one or more sequence CPUs are on the base unit, some of the I/O modules can be controlled by the sequence CPU and the others can be controlled by the F3RP61. It is recommended that the I/O module under the sequence CPU’s control be accessed indirectly by the IOC (F3RP61) through the internal devices of the sequence CPU. (See section 6 for more detail.)

Digital input modules of FA-M3 can interrupt CPUs upon a change of the state of the input signal. The BSP of F3RP61 has a function that transforms the interrupt into a message to a user-level process running on it. Based on this function, the device and driver support supports processing records upon an I/O interrupt.

2. Device type

In order to use the device and driver support, the device type (DTYP) field of the record must be specified to either “F3RP61” or “F3RP61Seq”. The former is for accessing the relays and registers of the I/O modules and the shared memory. The latter is for accessing internal devices (“D”, “I”, “B”) of the sequence CPUs on the same base unit.

3. Accessing I/O module

3.1. Accessing input relay (X)

Input relays are read-only devices. Binary input (bi) records, multi-binary input direct (mbbiDirect) records, long input (longin) and analog input (ai) records are supported by the device support. Three numbers, unit number, slot number. and the input relay number must be specified in the INP field as the following example shows.

```
record(bi, “f3rp61_example_1”) {
        field(DTYP, “F3RP61”)
        field(INP, “@U0,S2,X1”)
 }
```

The above record reads a value of either “on” (1) or “off” (0) from the first input relay (X1) of an I/O module in the second slot (S2) of the main unit (U0). The next example shows how to read 16 bits of status data on a group of input relays by using mbbiDirect records.

```
record(mbbiDirect, “f3rp61_example_2”) {
        field(DTYP, “F3RP61”)
        field(INP, “@U0,S2,X1”)
 }
```

In this case, the number in “X1” specifies the first relay to read. The status bits on X1 to X16 are read by the mbbiDirect record.

The next example shows how to read a digital value of 16 bits on a group of input relays by using longin records.

```
record(longin, “f3rp61_example_3”) {
        field(DTYP, “F3RP61”)
        field(INP, “@U0,S2,X1”)
 }
```

In this case, it is assumed that the Least Significant Bit (LSB) is on X1 and the Most Significant Bit (MSB) is on X16, and the value on the input relays is signed one. If the value is unsigned one, “&U” must follow the relay number as follows.

```
record(longin, “f3rp61_example_4”) {
        field(DTYP, “F3RP61”)
        field(INP, “@U0,S2,X1&U”)
 }
```

If the value is signed 32 bit one, “&L” must follow the relay number.

```
record(longin, “f3rp61_example_5”) {
        field(DTYP, “F3RP61”)
        field(INP, “@U0,S2,X1&L”)
}
```

Currently, unsigned 32 bit value is not supported.

The rule explained in the last three examples applies to ai records. The value read from the input relays goes into the raw value (RVAL) field of the ai record.

3.2. Accessing output relay (Y)

Output relays are read-write devices. Just replacing ‘X’ with ‘Y’ in the INP fields of the example records shown in the previous subsection suffices to read output relays.

In order to write them, binary output (bo) records, multi-bit binary output direct (mbboDirect) records, long output (longout) records, and analog output (ao) records can be used.

```
record(bo, “f3rp61_example_6”) {
        field(DTYP, “F3RP61”)
        field(OUT, “@U0,S2,Y1”)
}
```

The above records writes a value of either “on” (1) or “off” (0) onto the first output relay (Y1) of an I/O module in the second slot (S2) of the main unit (U0).

The next example shows how to write 16 bits of status data onto a group of output relays by using mbboDirect records.

```
record(mbboDirect, “f3rp61_example_7”) {
        field(DTYP, “F3RP61”)
        field(OUT, “@U0,S2,Y1”)
}
```

In this case, the number in “Y1” specifies the first relay to write. The status bits are written onto Y1 to Y16 by the mbboDirect record.

The next example shows how to write a value of 16 bits onto a group of output relays by using longout records.

```
record(longout, “f3rp61_example_8”) {
        field(DTYP, “F3RP61”)
        field(OUT, “@U0,S2,Y1”)
 }
```

In this case, it is assumed that the Least Significant Bit (LSB) goes onto Y1 and the Most Significant Bit (MSB) goes onto Y16, and the value is signed one. If the value is unsigned one, “&U” must follow the relay number as follows.

```
record(longout, “f3rp61_example_9”) {
        field(DTYP, “F3RP61”)
        field(OUT, “@U0,S2,Y1&U”)
 }
```

If the value is signed 32 bit one, “&L” must follow the relay number.

```
record(longout, “f3rp61_example_10”) {
        field(DTYP, “F3RP61”)
        field(OUT, “@U0,S2,Y1&L”)
 }
```

Currently, unsigned 32 bit value is not supported.

The rule explained in the last three examples applies to ao records. The value in the raw value (RVAL) field of the ao record is written onto the output relays.

3.3. Accessing data register

Analog I/O modules and other special modules, such as motion control modules, serial communication modules, etc., have many registers to hold the I/O data and relevant parameters. In order to read/write these registers, longin/longout, ai/ao records and mbbiDirect/mbboDirect records are supported. While some of the registers can be 32 bit ones, the device and driver support supports only reading/writing a value of 16 bits. 32 bit registers must be read/written, according to the specification of the I/O module, by using two records, one for the upper half and the other for the lower half of the 32 bit value.

The following example shows how to use longin records to read a 16 bit register.

```
record(longin, “f3rp61_example_11”) {
          field(DTYP, “F3RP61”)
          field(INP, “@U0,S3,A1”)
 }
```

The above record reads a value of 16 bit data from the first register (A1) of a module in the third slot (S3) of the main unit (U0).

The following example shows how to use longout records to write a 16 bit register.

```
record(longout, “f3rp61_example_12”) {
          field(DTYP, “F3RP61”)
          field(OUT, “@U0,S3,A1”)
 }
```

The above record writes a value of 16 bit data onto the first register (A1) of a module in the third slot (S3) of the main unit (U0).

The rule to specify an input (output) register address applies to ai (ao) records. The value read from (written onto) the device comes in (goes out) via the RVAL field of the ai (ao) records.

4. Handling special module

This section describes how to handle special modules that require some sequence logic to execute I/O operations. As an example, we consider a motion control module that controls the motion of a stepping motor by sending a train of pulses to the motor driver.

Suppose you drive a motor dedicated to an axis. In the first place, you make the F3RP61 set the number of pulses (distance) into a register of the motion control module by using longout or ao record(s). (Since the parameter is 32 bit long, two records are necessary to set the upper 16 bits and the lower 16 bits.)

Next, you make the F3RP61 turn on an output relay (EXE) to trigger the action. This can be performed by putting the value of one (1) into the VAL field of the following record. (The relay number varies from type to type. The following example records are just for illustration. See the manual of the module you use for the information. We assume the motion control module is in slot 4.)

```
record(bo, “f3rp61_motion_exe”) {
          field(DTYP, “F3RP61”)
          field(OUT, “@U0,S4,Y33”)
 }
```

The motion control module will respond to the command by turning on an input relay (ACK) if no error is found in the parameters given. The F3RP61 needs to check the ACK by monitoring the VAL field of the following record.

```
record(bi, “f3rp61_motion_ack”) {
          field(SCAN, “.1 second”)
          field(DTYP, “F3RP61”)
          field(INP, “@U0,S4,X1”)
 }
```

And then, the operation (sending pulses) starts. The F3RP61 is required to turn off the EXE after the ACK is turned on. The motion control module, then, turns off the ACK after the EXE is turned off.

When the operation completes, the motion control module informs the F3RP61 of the completion by turning on yet another input relay (FIN). Just like the ACK, the FIN needs to be checked by monitoring the VAL field of the following record.

```
record(bi, “f3rp61_motion_fine”) {
          field(SCAN, “.1 second”)
          field(DTYP, “F3RP61”)
          field(INP, “@U0,S4,X5”)
 }
```

Note that the records to monitor the input relays must be processed periodically.

The essential part of the EPICS sequencer program to manage the sequence described above in words will look like as follows.

```
ss exe_move {
        state wait_exe {
                when (f3rp61_motion_exe) {
                } state wait_ack
        }
        state wait_ack {
                when (f3rp61_motion_ack) {
                        f3rp61_motion_exe = FALSE;
                        pvPut(f3rp_motion_exe);
                } state_wait_fin
        }
        state wait_fin {
                when(f3rp61_motion_fine) {
                } state wait_exe
        }
}
```

The author reminds you that the above example is just for an illustration. The sequencer for the real operation involves a little bit more to handle many different types of commands of the motion control module, and to handle exceptions that can occur in the sequence (for example, an error caused by a wrong parameter set by the user).

5. Communication with sequence CPU

Two different types of methods are supported for an F3RP61 to communicate with a sequence CPU that works on the same unit. One method is based on the shared memory and the other is message-based communication. The former is fast access that finishes instantly, just like the access to I/O relays and registers of an I/O module. The latter is slow access that takes a few milliseconds of time to complete the transaction. For this reason, the device and driver support is separated into the two parts, the synchronous part ("F3RP61") for the former and the asynchronous part ("F3RP61Seq") for the latter.

5.1. Communication based on shared memory

The following is the basics to understand how the communication between F3RP61s and sequence CPUs works.

- Each CPU, an F3RP61 or a sequence CPU, has its own region.
- Any CPU, an F3RP61 or a sequence CPU, can write only its own region.
- Any CPU, an F3RP61 or a sequence CPU, can read any region.

In order to make the story simple, we consider the case where only one F3RP61 in slot 1 works with only one sequence CPU in slot 2 on the base unit. (See, Fig.1) From the rules mentioned above, how we can use the shared memory to make those two CPUs communicate each other is clear.

- If the data go from the F3RP61 (CPU1) to the sequence CPU (CPU2), use the area of the F3RP61 (CPU1), and vice versa.

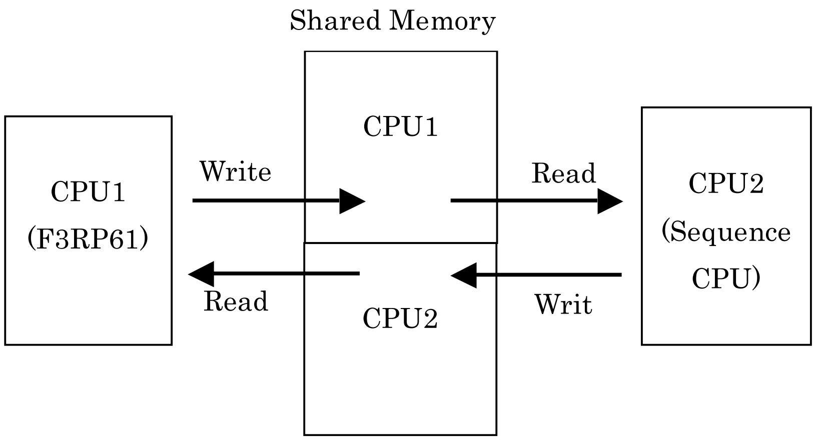

Fig. 1

While the sequence CPU can access the shared memory by bit (shared relay) or by word (shared register), the F3RP61 can access it only by word. With the aid of the ladder program development tool, WideField2, you can configure the shared memory area by specifying the number of shared relays and shared registers for each of the CPUs.

In the first place, we consider setting values from the F3RP61 to the ladder program running on the sequence CPU. The following record allows the F3RP61 to write the first word in the region, "CPU1" in Fig. 1.

```
record(longout, "f3rp61_example_13") {
        field(DTYP, "F3RP61")
        field(OUT, "@CPU1,R0")
}
```

Suppose the area, "CPU1", is configured as a set of shared relays, which is referenced by the ladder program running on the sequence CPU. When the F3RP61 writes a word into the area, the ladder program sees a write on 16 shared relays (E00001-E00016) occur all at once.

In contrast, if the area is configured as a set of shared registers, the ladder program sees a write on one shared register (R00001) occur. Note that the addressing manner is different between F3RP61's I/O space and "PLC device" referenced by ladder programs. While the former starts from zero (R0), the latter starts from one (R00001).

If the area is configured as both shared relays and shared registers, the former comes first and the latter follows them. For example, supposed the area, "CPU1" includes two words of relays (32 relays) and four words of shared registers. The address map looks like as follows.

R0:	E00001 – E00016
R1:	E00017 – E00032
R2:	R00001
R3:	R00002
R4:	R00003
R5:	R00004

Next, we consider reading values written by the sequence CPU. By using the following record, the F3RP61 can read the first word from the region, "CPU2" in Fig. 1, which starts form just after the last word in the above address map, i.e., R6.

```
record(longin, “f3rp61_example_14”) {
          field(DTYP, “F3RP61”)
          field(INP, “@CPU2,R6”)
}
```

What the F3RP61 sees varies depending on how the area is configured. It can be 16 bits of shared relays or one word of shared register as explained above.

You might think that there is no reason to specify the CPU number in the INP field if the shared memory is a flat space being addressed by a series of sequential address numbers. However, we do need to specify the CPU number as shown in the above example. We do not discuss it any more since it is a bit complicated story. See the relevant manuals of F3RP61 for more details.

Other record types, ai/ao and mbbiDirect/mbboDirect, are also supported to read/write the shared memory.

5.2. Accessing internal device of sequence CPU

The alternative for an F3RP61 to communicate with a sequence CPU is to use the message-based transaction supported by the kernel-level driver. The following example shows how to set (1)/ reset (0) an internal relay (‘I’) of a sequence CPU in slot 2.

```
record(bo, “f3rp61_example_15”) {
          field(DTYP, “F3RP61Seq”)
          field(OUT, “@CPU2,I4”)
}
```

Note that the device type must be “F3RP61Seq” in this case. In order to read back the result, you can use the following record.

```
record(bi, “f3rp61_example_16”) {
          field(DTYP, “F3RP61Seq”)
          field(INP, “@CPU2,I4”)
}
```

In order to write a data register (‘D’) of the sequence CPU, the following record can be used.

```
record(longout, "f3rp61_example_17") {
        field(DTYP, "F3RP61Seq")
        field(OUT, "@CPU2,D7")
}
```

Again, the result can be read back by using the following record.

```
record(longin, "f3rp61_example_18") {
        field(DTYP, "F3RP61Seq")
        field(INP, "@CPU2,D7")
}
```

Other record types, ai/ao and mbbiDirect/mbboDirect, are also supported to read the data registers. Another internal device, file registers ('B'), can be read/written by using those record types. Accessing other internal devices than 'I', 'D' and 'B' is not currently supported.

6. A caution in using F3RP61 and sequence CPU side-by-side

This section gives you a caution in using an F3RP61 to monitor the relay status of the I/O modules that are used by a sequence CPU on the same base unit.

The point is rebooting the F3RP61 can stop the ladder program running on the sequence CPU if the F3RP61 has had a direct access to the I/O module for the reason described below.

When an I/O module is accessed by the F3RP61, the I/O module recognizes and remembers that the F3RP61 is its master. When the Linux system on the F3RP61 is rebooted, the F3RP61 broadcasts the fact by using a signal on the PLC-bus. The I/O modules that recognize the F3RP61 as their master reset themselves when they detect the signal. This makes the I/O modules un-accessible by the sequence CPU and makes the ladder program stop with an I/O error, if the I/O modules are used by the sequence CPU.

For this reason, it is preferable to read the relay status indirectly via some internal devices (‘I’, ‘D’, ‘B’) of the sequence CPU or the shared memory (‘E’, ‘R’) by using one of the methods described in the previous section.

7. I/O interrupt support

Digital input modules of FA-M3 can interrupt CPUs when they detect a rising edge or falling edge of the input signals. The kernel-level driver can transform the interrupt into a message to a user-level process. Based on the function, processing records by I/O interrupt is supported. Any records that have the DTYP field value of “F3RP61” and the SCAN value of “I/O Intr” get processed upon an interrupt on a specified channel of the specified module. This feature allows you to trigger a read/write operation by external trigger.

Suppose a unit comprised of an F3RP61 (slot 1), a digital input module (slot2) and an A/D module (slot 3). The following record reads the first data register (A1) of the A/D module upon a trigger input onto the first channel (X1) of the digital input module. (The INP field specification is “@[I/O data channel]:[interrupt source]”.)

```
record(ai, “f3rp61_example_19”) {
          field(DTYP, “F3RP61”)
          field(SCAN, “I/O Intr”)
          field(INP, “@U0,S3,A1:U0,S2,X1”)
}
```

If you add a D/A module in slot 4, you can write output data into the first data register (A1) of the D/A module upon the same interrupt by using the following record.

```
record(ao, “f3rp61_example_20”) {
          field(DTYP, “F3RP61”)
```

```
                field(SCAN, “I/O Intr”)
                field(OUT, “@U0,S4,A1:U0,S2,X1”)
        }
```

The following example might seem a little bit strange, but it helps you see how quick F3RP61 can respond to interrupts.

```
        record(bi, “f3rp61_example_21”) {
                field(DTYP, “F3RP61”)
                field(SCAN, “I/O Intr”)
                field(INP, “@U0,S2,X1:U0,S2,X1”)
        }
```

This bi record reads the status of the input relay into which the trigger signal goes. Suppose the trigger signal is a pulse and the digital module generates an interrupt at the rising edge. If the bi record, which gets processed by the interrupt, can read the status before the signal level falls down to low, the record reads “high” (1). Otherwise, it reads “low” (0). By changing the pulse duration with checking the record value, you can see how long it takes for the record to get processed starting from the interrupt.