MELSEC iQ-R C 言語インテリジェント機能ユニット上への EPICS 組み込みと Python によるデバイスサポートの検証

EVALUATION OF EMBEDDED EPICS WITH Python-BASED DEVICE SUPPORT ON MELSEC iQ-R C INTELLIGENT FUNCTION MODULE

内山暁仁#, 込山美咲, 山田一成
Akito Uchiyama #, Misaki Komiyama, Kazunari Yamada
RIKEN Nishina Center

Abstract

The RIBF control system is based on EPICS and connects to many TCP/IP-based devices, both commercial and in-house. However, unexpected power loss of devices or network switches can break socket communication, and reconnection is often not established, leading to operational issues. Mitsubishi Electric's MELSEC series PLCs are also employed in some cases, where EPICS IOCs typically communicate with the PLC over TCP/IP from a separate Linux machine. To improve communication reliability in such configurations, we tested embedding EPICS IOC directly on the MELSEC iQ-R "C Intelligent Function Module", which runs Linux. As this controller requires native compilation, Python was used for initial device support development, utilizing PyDevice to interface with EPICS. This paper reports on the development approach, implementation status, and future prospects of Python-based EPICS device support on the MELSEC iQ-R platform.

1. はじめに

RIBF（RIKEN RI Beam Factory）制御系は、EPICS（Experimental Physics and Industrial Control System）をベースに構築されている。RIBF 稼働当初より、電磁石電源は VME バスに搭載された VME CPU 上の EPICS IOC（Input/Output Controller）が、デバイスと VME バスを介して接続され、カーネルレベルのデバイスドライバを通じて制御が行われている[1]。EPICS R3.14 系の導入に伴い、電磁石電源以外のデバイスの EPICS IOC との接続はバスベースだけでなく Ethernet ベースのシステムも導入された。現在 N-DIM と呼ばれる独自開発のデバイスをはじめ、MELSEC、OMRON、FA-M3 といった市販 PLC や、各種計測器が Socket で接続され、NetDev[2] や AsynDriver[3]、StreamDevice[4]等の EPICS extensions を用いたデバイスサポートにより制御されている[5]。

Ethernet は OSI 参照モデルにおける物理層およびデータリンク層の技術として広く普及しており、汎用的かつ扱いやすいインタフェースであるが、EPICS IOC と TCP/IP ベースのデバイスとの通信は非同期な Socket 接続で行われるため、デバイスやスイッチの予期せぬ電源断・再起動により通信が切断された後、自動的に再接続されないという問題がある。このため、制御系の安定運用を妨げる要因となっていた。このような課題に対して、RIBF 制御系では横河電機製 FA-M3 PLC に Linux CPU である F3RP61/F3RP71 を搭載し、EPICS を組み込み、直接実装させる構成が多く採用されてきた[6-8]。EPICS が PLC と同一のバス上に存在することで、前述の通信切断問題を回避できることが利点である。

一方で、FA-M3 ではなく、産業界で広く使用されている三菱電機製の PLC（MELSEC シリーズ）を RIBF 制御系に導入するケースもある。その場合、従来は別の Linux マシン上で動作する EPICS IOC から MC プロトコルと TCP/IP を介した非同期なデバイスサポートにより PLC にアクセスする構成を取っていた[1, 9-11]。

本研究では、このような構成においても EPICS IOC と制御デバイス間の通信信頼性を向上させることを目的に、Linux が動作する MELSEC iQ-R シリーズの PLC モジュールである C 言語インテリジェント機能ユニット（RD55UP06-V / RD55UP12-V）上で、EPICS を動作させる実証試験を実施した。

a-uchi@riken.jp

2. C 言語インテリジェント機能ユニット

2.1 特徴

MELSEC iQ-R シリーズにおいて、Linux が動作する C 言語インテリジェント機能ユニット（RD55UP06-V / RD55UP12-V）は、シーケンサ CPU ユニット（例：R08CPU）と組み合わせて使用する構成となっており、単体では動作しない。本ユニットは、シーケンサ CPU との間でバッファメモリを介してデータの受け渡しを行うことで、PLC デバイスへのアクセスや I/O 制御を実現している。標準的な機能のユニットと同様にベースユニットに装着して使用する形式であり、スロット位置に特別な制約はなく、他のユニットとの干渉を気にせずに柔軟な構成が可能である。

また、本ユニットには Debian ベースの Linux が搭載可能であり、ユーザは C 言語や Python を用いて自由にアプリケーションを開発可能である。Linux に標準で備わるツールチェーンや各種ライブラリをそのまま利用できるため、EPICS をはじめとする制御系フレームワークの導入も比較的容易である。

PLC 側との通信においては、CITL（C 言語インテリジェント機能ユニット専用関数）ライブラリを用いた MDR（MELSEC Datalink Library for iQ-R）通信、バッファメモリアクセス方式に対応しており、リアルタイム性が求めら

れる産業用制御用途においても、十分な応答性能が期待できる。

2.2 仕様

Table 1 に C 言語インテリジェント機能ユニットのハードウェア仕様を示す[12]。大きな違いは RD55UP06-V はメモリ 128 MB と RD55UP12-V は 1 GB という事である。当初、RD55UP06-V で開発を行っていたが、メモリ不足で OOM killer が走る事があるため、ルートパーティションに 4 GByte の swap file を作成する事で対応した。Figure 1 にデバイスサポート開発中におけるインストールされた RD55UP12-V の様子を示す。

Table 1: Hardware Specifications of C Intelligent Function Modules

	RD55UP06-V	RD55UP12-V
CPU	ARM® Cortex®-A9 Dual Core	ARM® Cortex®-A9 Dual Core
Memory	128 Mbyte	1 Gbyte
Ethernet port number	1	2

Figure 1: The C Intelligent Function Module RD55UP12-V, which was installed in the test environment, is mounted in the rightmost slot of the base unit.

2.3 アクセス方式

C 言語インテリジェント機能ユニットは、シーケンサ CPU との間でアクセスライブラリ(CITL)を介した通信によって、PLC モジュールへのアクセスを実現する[13]。アクセス方式としては、大きく分けて以下の 2 種類が用意されている。ひとつは MDR 通信方式であり、C 言語アプリケーションから CITL ライブラリの mdrSend および mdrReceive 関数を用いて、バッファメモリ経由でコマンドベースの要求を行うものである。これは、デバイス単位での読み書きに適しており、アクセス粒度が細かい制御に向いている。もうひとつはバッファメモリアクセス方式である。CITL_ToBuf / CITL_FromBuf 関数を用いて共有されたデータ領域を用いて、あらかじめ定義されたオフセットに直接データを読み書きする形式である。この方式は、周期的なデータ交換や複数変数の一括処理に適しており、高速な定周期制御用途に向いている。

本研究では、まずは挙動確認および基本的な機能の実装を目的として、mdrSend および mdrReceive 関数を用いた MDR 通信方式に限定して開発を進めた。基本的な MDR 通信方式でレジスタにアクセスするプログラム例を Listing 1 に示す。

Listing 1: Example Python Program that Reads and Displays the Value of Register D100

```
#!/usr/bin/python3
from CITL_LinuxPy import *

def main():
    # 初期化
    mdrAppInit()

    # 通信チャネル =12(Bus アクセス), 号機番号 =0(明示指定), タイムアウト =100ms
    lPath = []
    ret = mdrOpen(12, 0, lPath, 10)
    if ret != 0:
        print(f"[ERROR] mdrOpen failed, ret = {ret}")
        return
    print(f"[INFO] mdrOpen success, lPath = {lPath[0]}")

    # D 読み出し
    dev_code = 13       # D レジスタ
    addr = 100
    lSize = [2]         # 1 ワード
    sData = [0]

    ret = mdrReceive(lPath[0], 4, 0, 0, 0, 0, dev_code, addr, lSize, sData)

    if ret == 0 and sData:
        val = sData[0] & 0xFFFF   # 「符号なし 16bit 整数」に変換
        print(f"[INFO] D{addr} = {val}")
    else:
        print(f"[ERROR] mdrReceive failed, ret = {ret}, sData = {sData}")

    # 終了処理
    mdrClose(lPath[0])
    print("[INFO] mdrClose done")

if __name__ == "__main__":
    main()
```

3. ビルド環境と実行

3.1 Linux

本ユニットには、Debian GNU/Linux 10.11(Buster)をベースとした組込み Linux が動作している。OS イメージはサポート Web サイト[14]より提供されており、これを SD カードに書き込むことで、当該カードからの起動が可能となる構成である。ファイルシステムは標準的な Linux のディレクトリ構成に準拠しており、CITL ライブラリは /usr/lib/citl/以下に配置されている。

本ユニットはクロス開発環境を必要とせず、ネイティブ開発が可能であるため、ユーザアプリケーションの配置および開発は/root/ディレクトリ下で行っている。開発環境として、apt パッケージ管理システムを利用し、gcc や make などのツールチェーンを導入することで、ユニット上での直接ビルドを実現している。

また、起動後は標準搭載のイーサネットインタフェース

を介して SSH によるリモートログインが可能であり、通常の Debian システムと同様にコマンドラインベースでの操作が行える。

3.2 PyDevice を用いたデバイスサポート

EPICS IOC の動作試験およびデバイスサポートの設計、実装にあたっては、本ユニット上で C 言語によるコードを都度 make してビルドを行う開発スタイルは、CPU 性能や開発環境の制約から効率的とは言い難い。そこで、C 言語で提供されているライブラリを Python からバインディング経由で呼び出すことが可能である点に着目し、まずはインタプリタ型言語である Python を用いてデバイスサポートの初期開発を行うこととした。

Python による開発には、従来から利用実績のある PyDevice[15, 16]フレームワークを採用した。PyDevice は、Python で EPICS のデバイスサポートを実装するための仕組みであり、比較的容易に機能の試行錯誤が可能であることから、本研究におけるプロトタイプ開発の段階で有用であると判断した。

3.3 開発環境

今回の開発環境で利用したバージョンを Table 2 に示す。EPICS 本体に加えて、Sequencer、設定値の自動保存（AutoSave）、および Python ベースのデバイスサポート開発に用いた PyDevice のバージョンを明記している。各コンポーネントの互換性を保つため、RIBF 制御系で安定運用実績のあるバージョンを選定した。

Table 2: Software Versions Used in the Development Environment

Package	Version
EPICS base	base-3.15.9
Sequencer	R2.2.9
AutoSave	R5-11
PyDevice	R1.2.1
Python	3.7.3

4. デバイスサポートの検討

4.1 読み出しテスト

CITL ライブラリを用いてシーケンサ CPU のレジスタを読み出す処理は、一般に、初期化（mdrAppInit）に続いて、通信経路の確立（mdrOpen）、コマンドの送信（mdrSend）、応答の受信（mdrReceive）、そして経路の解放（mdrClose）という一連の関数呼び出しによって構成される[12]。本節で、通信要求毎に mdrOpen と mdrClose を毎回実行する場合と、mdrOpen により確立した通信経路を維持したまま繰り返し mdrSend / mdrReceive を実行する場合とで、それぞれのレジスタ読み出しにかかる時間をシンプルなプログラムを作成し比較測定した。

毎回 mdrOpen および mdrClose を実行して D レジスタを 100 個読み出すテストを行った結果、全体で約 7.0 sec を要し、1 レジスタあたり約 70 msec の処理時間となった。これに対し、mdrOpen / mdrClose を 1 回のみ実行し、その接続を維持したまま D レジスタを 100 個読み出す場合には、合計で 0.656 sec となり、1 レジスタあたり約 6.6 msec と高速化された。このことから、運用においては mdrOpen による通信経路を維持し、定周期でレジスタを読み書きする方式が望ましいと判断される。

さらに、複数コネクションにおける性能を確認するため、D レジスタを 1,000 個読み出す処理を行うスレッドを 2 つ並列に実行するテストも行った。その結果、各スレッドともに処理時間は約 10.6 sec であり、1 レジスタあたりの読み出し時間は約 10 msec となった。この結果から、複数コネクションの同時使用は技術的には可能であるが、単体での接続に比べて明確な性能向上は見られず、実用性は限定的であると考えられる。

4.2 通信経路のスレッド制約と対応方式

CITL ライブラリを用いた MDR 通信では、mdrOpen により取得される通信経路ハンドル（lPath）は、ハンドルを取得したスレッド内でのみ使用する必要があり、スレッド間で共有しての使用はできない。これは仕様上の制約であるため、mdrOpen ～ mdrClose による一連の通信処理はすべて同一スレッド内で完結させる必要がある。つまり、複数のスレッドやプロセス間で lPath を共用する構成は採用できないため、以下の 2 方式を検討した。一つはローカルソケット方式であり、これはレジスタアクセス処理を独立したプロセスとして分離し、EPICS 側とは UNIX ドメインソケットによるプロセス間通信（IPC）を行う構成である。プロセスごとに lPath を独立に保持できる利点がある。もう一つはワーカースレッド方式である。これは単一プロセス内でレジスタアクセス専用のワーカースレッドを起動し、メッセージキュー等でスレッド間通信を行う構成になっている。mdrOpen をスレッド内に閉じ込めて管理できるため、効率的かつスレッドセーフに処理できる。

4.3 ローカルソケット方式

ローカルソケット方式では、mdrOpen～mdrClose を含む MDR 通信処理を別プロセスとして分離し、EPICS IOC 側のプロセスとは UNIX ドメインソケットを介して通信を行う。MDR 通信プロセスは、ソケットを通じて受信した要求に応じてシーケンサ CPU のレジスタ読み書きを行い、その結果を応答として返送する。

この方式の利点は、通信処理がプロセスとして独立しているため、デバイスサポート側はあたかも通常のソケット通信デバイス（たとえば asynDriver や StreamDevice）であるかのように実装できる点にある。つまり、EPICS 側はソケット経由でデータを送受信するだけでよく、低レベルな PLC アクセスの詳細は切り離して管理する事ができる。一方で、IOC 起動時にデータベースファイル(.db)よりも先に通信プロセスを起動する必要があるため、スタートアップスクリプト(st.cmd)の記述順や運用設計に若干の注意が必要である。この点が運用面での導入のしづらさとして認識された。

実際にはこの方式を採用しなかったものの、モジュールの疎結合性や柔軟性の面からは有力なアーキテクチャであり、今後の保守性やマルチユニット対応といった観点での再評価も考慮されるべきである。

4.4 ワーカースレッド方式

EPICS IOC プロセス内に MDR 通信専用のスレッド（ワーカースレッド）を生成し、他のスレッドからの要求を逐次処理するワーカースレッド方式でのデバイスサポートの開発を行った。Figure 2 に示すように、EPICS IOC における record 処理が PyDevice ベースのデバイスサポートに到達すると、read/write の要求はキューに詰められ、バックエンドで動作するワーカースレッドに転送される。ワーカースレッドは、要求内容に応じて CITL ライブラリの mdrSend および mdrReceive 関数を順次呼び出し、シーケンサ CPU との通信を実行する構成となっている。

MDR 通信に必要な初期化（mdrAppInit）および通信経路の確立（mdrOpen）はワーカースレッド内で一度だけ行い、その後のレジスタアクセスもすべて同一スレッド内で完結するよう設計している。これにより、CITL ライブラリの仕様に則った安全な通信処理が可能となっている。実装には POSIX スレッドと条件変数を用い、スレッド間での非同期要求の通知と応答待機を実現している。また、読み出し要求と書き込み要求を分類し、書き込み要求を優先的に処理する優先度付きの処理ロジックも導入している。これにより、制御信号などの応答性が重視される用途において、書き込み遅延の回避が図られている。開発されたデバイスサポートを使ったデータベースの例を Listing 2 に示す。

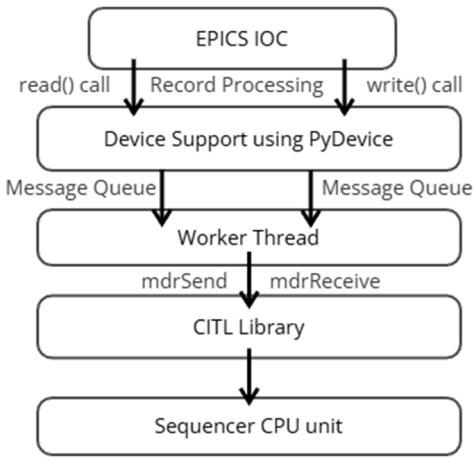

Figure 2: Overall architecture of the device support using PyDevice. Read and write requests from EPICS IOC are handled by the device support layer, which pushes requests to a message queue. A worker thread receives the requests and communicates with the Sequencer CPU unit via the CITL library functions (mdrSend, mdrReceive).

5. 結果と課題

ワーカースレッド方式は、ローカルソケット方式と異なり、単一プロセス内ですべての処理が完結するため、st.cmd ファイル内における外部プロセスの起動順序管理が不要であり、運用上の利便性に優れる。また、EPICS のデバイスサポートとの統合性も高く、将来的な保守性や機能拡張の面でも有利であると判断し、本方式による実装を採用した。実際の運用では、PyDevice を用いたワーカースレッド方式により、標準的なレコード型である bi/bo、longin/longout、waveform を用いた D レジスタおよび M レジスタの制御を問題なく実現できた。

一方で、PyDevice をデバイスサポートとして用いる場合、DTYP フィールドの定義が”pydev”のような記述に限定され、DTYP を見ただけでは MELSEC 用であることが明示されないという課題がある。また、mbbiDirect/mbboDirect レコードがサポートされていないことも、本方式の制約として挙げられる。今後はこれらの課題を解消することを目的として、有用性の検証ができたワーカースレッド構成を維持しつつ、より柔軟な拡張が可能な asynDriver ベースのデバイスサポートの開発を進め、最終的に RIBF 制御系への本格導入を目指す予定である。

Listing 2: Example EPICS Database File Using PyDevice for MELSEC iQ-R C Language Intelligent Function Module

```
# D100 レジスタ（ワード）読み取り
record(ai, "plc:d100:ai") {
      field(DTYP, "pydev")
      field(INP, "@MelThread.read('D100')")
      field(SCAN, "1 second")
}

# M100 レジスタ書き込み
record(bo, "plc:m100:bo") {
      field(DTYP, "pydev")
      field(OUT, "@MelThread.write('M100',VAL)")
}

# M96 レジスタ（ワード）書込み
record(longout, "plc:m96:longout") {
      field(DTYP, "pydev")
      field(OUT, "@MelThread.writeW('M96',VAL)")
}

# D100 レジスタ（waveform）読込み
record(waveform,"plc:d100:wave"){
      field(DTYP, "pydev")
      field(INP, "@MelThread.read('D100',10)")
      field(FTVL, "LONG")
      field(NELM, "10")
      field(SCAN, "1 second")
}
```

参考文献

[1] M. Komiyama *et al.*, “Status of Control System for RIKEN RI-Beam Factory”, in Proc. ICALEPCS'07, Oak Ridge, TN, USA, Oct. 2007, paper TPPB11, pp. 187-189.

[2] J. Odagiri *et al.*, “EPICS Device/Driver Support Modules for Network-based Intelligent Controllers”, in Proc. ICALEPCS'03, Gyeongju, Korea, Oct. 2003, paper WP563, pp. 494-496.

[3] M. R. Kraimer, M. Rivers, and E. Norum, “EPICS Asynchronous Driver Support”, in Proc. ICALEPCS'05, Geneva, Switzerland, Oct. 2005, paper P3_074.

[4] StreamDevice, https://github.com/paulscherrerinstitute/StreamDevice

[5] M. Komiyama *et al.*, “Current Status of the Control System for the RIKEN Accelerator Research Facility”, in Proc. ICALEPCS'03, Gyeongju, Korea, Oct. 2003, paper MP528, pp. 107-109.

[6] M. Komiyama, M. Fujimaki, N. Fukunishi, J.-I. Odagiri, and A. Uchiyama, “Upgrading the Control System of RIKEN RI Beam Factory for New Injector”, in Proc.

ICALEPCS'09, Kobe, Japan, Oct. 2009, paper TUP084, pp. 275-277.

[7] A. Uchiyama, M. Fujimaki, N. Fukunishi, and M. Komiyama, "Integration of Standalone Control Systems into EPICS-Based System at RIKEN RIBF", in Proc. PCaPAC'16, Campinas, Brazil, Oct. 2016, pp. 35-37. doi:10.18429/JACoW-PCAPAC2016-WEPOPRPO12

[8] A. Uchiyama *et al*., "Design of Reliable Control with Star Topology Fieldbus Communications for an Electron Cyclotron Resonance Ion Source at RIBF", in Proc. PCaPAC'18, Hsinchu City, Taiwan, Oct. 2018, pp. 105-108. doi:10.18429/JACoW-PCaPAC2018-WEP30

[9] A. Uchiyama *et al*., "Control System of the SRILAC Project at RIBF", in Proc. ICALEPCS'21, Shanghai, China, Oct. 2021, pp. 147-152. doi:10.18429/JACoW-ICALEPCS2021-MOPV015

[10] M. Nishimura *et al*., "Upgrade of the HYPER ECR Ion Source Control System for Integration into the RIBF Control System", Proceedings of the 13th Annual Meeting of Particle Accelerator Society of Japan, August 8-10, 2016, Chiba, Japan, pp. 660-663.

[11] A. Uchiyama *et al*., "Development of Embedded System for Running EPICS IOC by Using Linux and a Single Board Computer", in Proc. ICALEPCS'07, Oak Ridge, TN, USA, Oct. 2007, paper WPPA09, pp. 334-336.

[12] MELSEC iQ-R C 言語インテリジェント機能ユニットユーザーズマニュアル（スタートアップ編）.

[13] MELSEC iQ-R C 言語インテリジェント機能ユニット Linux スタートアップマニュアル.

[14] https://www.timesys.com/mitsubishi-CITL-linux-registration/

[15] PyDevice, https://github.com/klemenv/PyDevice

[16] A. Uchiyama *et al*., "Introduction of Ethernet-based field networks to inter-device communication for RIBF control system", in Proc. ICALEPCS'23, Cape Town, South Africa, Oct. 2023, pp. 1384-1387. doi:10.18429/JACoW-ICALEPCS2023-TUPDP050