SMPTE ST 2110のソフトウェア実装事例
～ GStreamerによる実装 ～

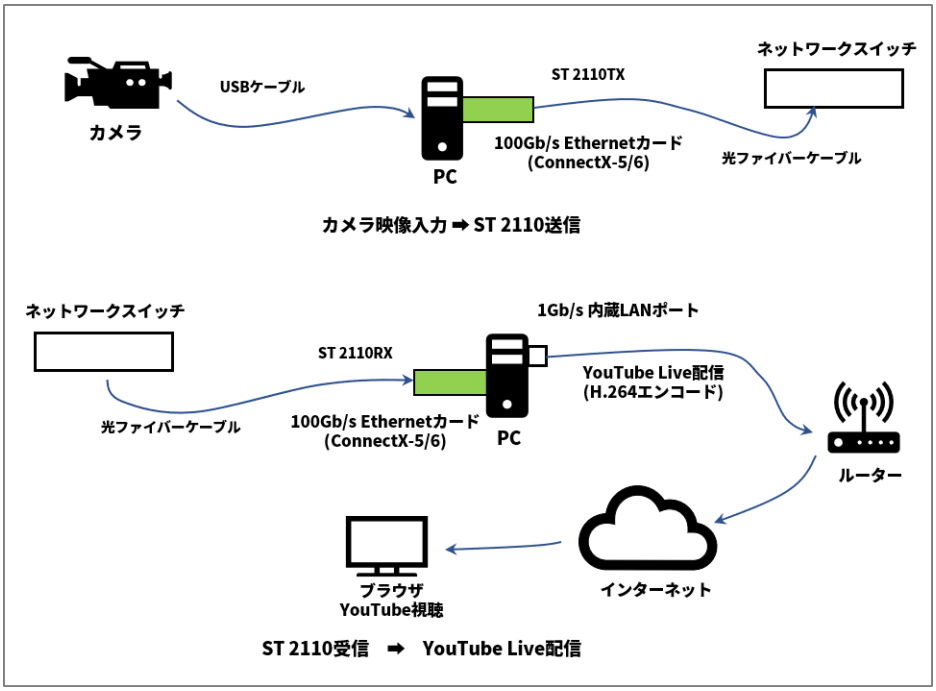
私たちの生活を支えるテクノロジーは、日々目覚ましい進化を続けています。その恩恵を受けて、私たちの日常生活は豊かになってきています。メディア&エンターテインメント(M&E)とよばれる、放送を中心とした世界においても例に違わず、現在進行形で大きな進化を遂げようとしています。そんな中、IPライブ伝送規格「SMPTE ST 2110」をソフトウェアで実装するニーズが高まっています。ソフトウェアで実装することにより、開発コストの削減は元より、既存のオープンソース技術との親和性が高まります。そこで今回は、「SMPTE ST 2110のソフトウェア実装事例」と題して、GStreamerによってSMPTE ST 2110をソフトウェアで実装する様子を解説していきます。

[SMPTE ST 2110のソフトウェア実装事例]
第1章　基本の構成
第2章　SMPTE ST 2110をソフトウェアで実装

第1章　基本の構成

GStreamerの豊富な機能を利用することで、カメラ入力映像をリアルタイムにST 2110で送信したり、ST 2110で受信したコンテンツをYouTubeにライブ配信が出来ます。（図表1）

100Gb/sのイーサネットカードにはNVIDIAのConnectX-5/6を使用しました。また、ConnectX-5/6の制御ソフトウェアとしてNVIDIAが提供するRivermaxを利用しています。



図表1

ConnectX-5/6 & Rivermax

ConnectX-5/6はNVIDIAが提供する100Gb/sのPCIeイーサネットカードで、ST 2110オフロード機能を備えています。Rivermaxは同社が提供するストリーミングIPライブラリーです。RivermaxとConnectX-5/6の組み合わせで、カーネルバイパス機能を使ってST 2110パケットの送受信を効率的に行うことができます。

GStreamer

▶ 映像・音声アプリケーションとして

Rivermaxはストリーミング IPパケットを送受信するライブラリーなので、ST 2110で受信した映像をモニターに表示したり、カメラから取り込んだ映像をST 2110で送信したりするには、それら映像処理を行うためのアプリケーションが別途必要となります。今回は映像・音声アプリケーションとしてGStreamerを使いました。

▶ Rivermax用 GStreamerエレメントの作成

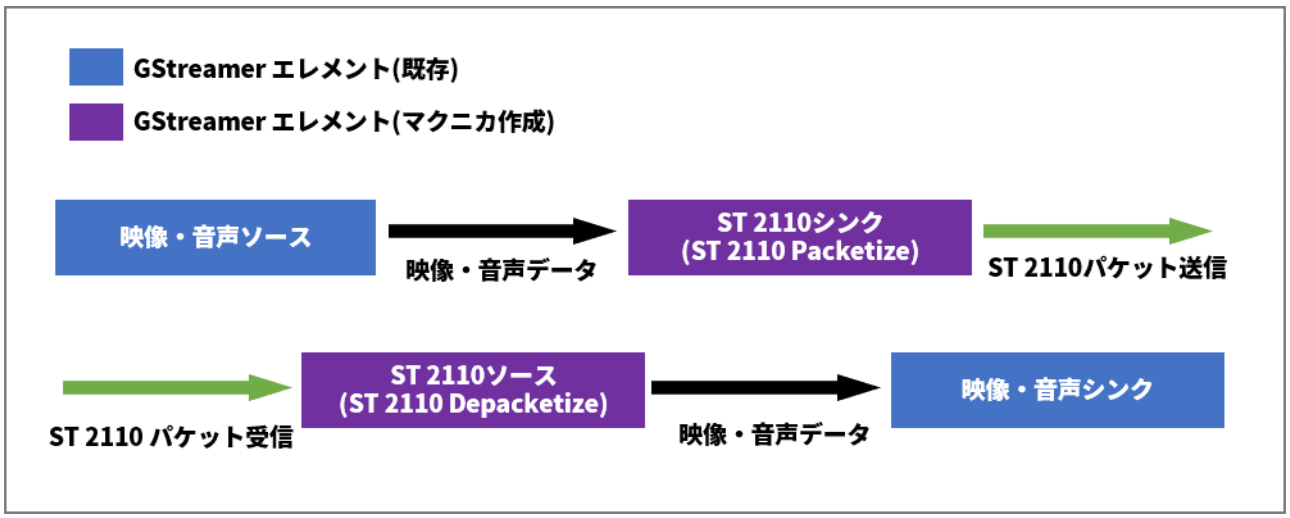
Rivermaxはカーネルバイパス機能を使ってST 2110パケットの送受信を効率的に行います。そのため上位アプリケーションはソケットプログラミングではなくRivermax APIでIPパケットのやり取りを行います。GStreamerにはUDPの送受信を行うエレメントが存在しますがソケット経由での送受信となるので、このエレメントではRivermaxのAPIを制御できません。

また、Packetize／Depacketize（RTP,ST 2110ヘッダーの付与／解析)はRivermaxより上位のアプリケーションで行う必要があります。GStreamerにはRTPパケットのPacketize／Depacketizeを行うエレメントも存在しますが、ST 2110には対応していません。

そこで、ST2110のPacketize／Depacketizeを行い、Rivermax API経由でIPパケットのやり取りを行うGStreamerエレメントを作成しました。

この作成したエレメントは、TX（GStreamer用語ではSink)方向であれば、前段のGStreamerエレメントからx-raw（I420,BGRx等)データを受け取り、YCbCr形式でST 2110にPacketizeして、Rivermax APIに渡すことでST 2110送信を行います。逆にRX(GStreamer用語ではSrc)方向であれば、YCbCr形式のST 2110パケットを受信すると、Rivermax APIから読みだしてDepacketizeし、x-raw(I420,BGRx等)データで後段のGStreamerエレメントに渡します。

これら2種類のエレメントを作成するだけで、後はGStreamerの豊富な機能が利用可能となります。（図表2）



図表2

実現できた機能

自作エレメント＋GStreamer既存エレメントの組み合わせで、以下の機能が実現できました。自分でコードを書いたのは上記自作エレメントだけで、あとは既存のエレメントの組み合わせをコマンドラインから叩くだけです。

[Video TX]
カメラ映像入力 => ST 2110 TX
H.264 Realtime Decode => ST 2110 TX
Color bar生成 => ST 2110 TX

[Video RX]
ST 2110 RX => YouTube Live配信
ST 2110 RX => モニター表示
ST 2110 RX => H.264 Realtime Encode

[Audio TX]
マイク入力 => ST 2110 TX
AAC/MP3 Realtime Decode => ST 2110 TX
Beep音生成 => ST 2110 TX

[Audio RX]
ST 2110 RX => スピーカー出力
ST 2110 RX => MP3 Realtime Encode
ST 2110 RX => スペクトラムアナライザをモニター表示

映像は2K/4K、50Hz/59.94Hzで動作できています。複数Streamを同時実行可能です。ConnectX-5/6の帯域的には100Gb/sまで流せますが、この構成で実際に何Streamまで同時実行可能なのでしょうか。性能評価については次回以降行っていきます。

1章 ー おわりに

いかがだったでしょうか。第1章では「SMPTE ST 2110」をGStreamerで扱うために知っておきたい基本的な構成についてお話ししました。
次の2章では、マクニカ作成GStreamerエレメントやGStreamerコマンドラインの詳細、そして性能評価についてご説明します。

第2章　SMPTE ST 2110をソフトウェアで実装

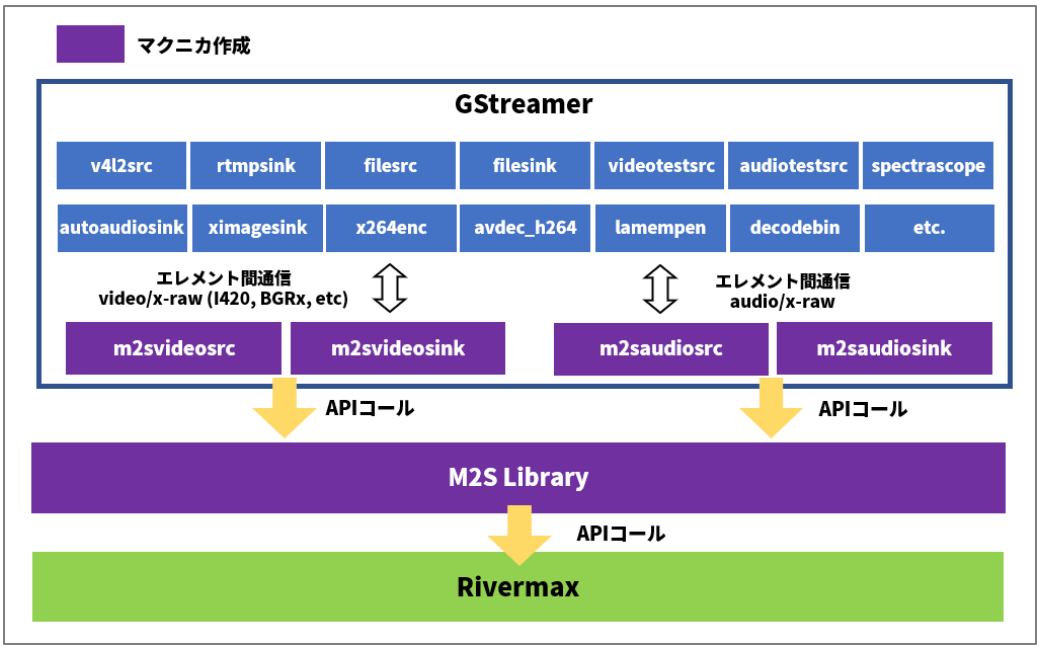
第1章では、IPライブ伝送規格「SMPTE ST 2110」をGStreamerで扱うための基本的な構成をお話しました。第2章では、マクニカ作成GStreamerエレメントの詳細および、GStreamerコマンドラインの実例をご紹介します。

マクニカ作成 GStreamerエレメントの構成

ST 2110に関する処理のコードをGStreamerエレメントに直接記述するのではなく、下位層となるライブラリーを作成し、GStreamerエレメントからはこのライブラリーを呼び出す構成としました。GStreamerエレメント内部はGStreamer特有の処理に特化し、メインとなる処理はライブラリー側に実装することで、このライブラリをGStreamer以外の映像・音声処理アプリケーションで利用することも可能となります。このライブラリーをM2S Library (Macnica Media Streaming Libraryの略) と命名しました。

GStreamerを使わない例として、FFMPEGで処理した映像データをM2S Libraryに渡すことでST 2110送信をする、といったアプリケーションを実装することも可能です。

GStreamerエレメントはそれぞれ、m2svideosrc (Video Src)、m2svideosink (Video Sink)、m2saudiosrc (Audio Src)、m2saudiosink (Audio Sink)と命名しました。（図表3）



図表3

各ブロックの役割は以下のようになります。

[GStreamer]　映像・音声処理
[M2S Library]　フォーマット変換、ST 2110 Packetize/Depacketize
[Rivermax]　ST 2110送受信

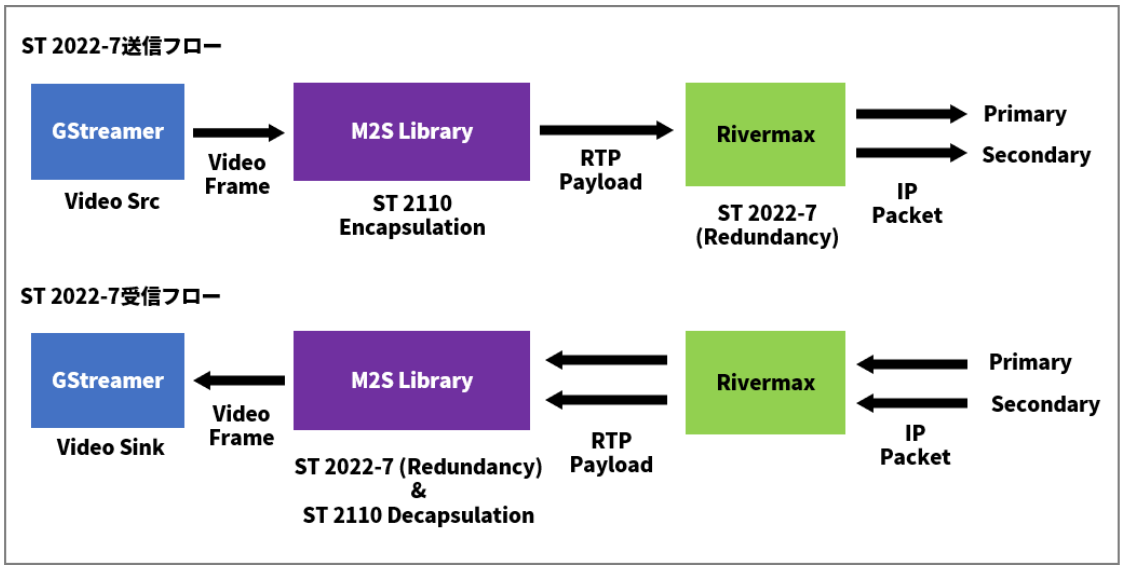
M2S LibraryのフォーマットにはCUDAによるGPUプログラミングを利用しています。フォーマット変換はピクセルごとの処理になるため、CPUでのシーケンシャルなループ処理では負荷が大きくなります。一方GPUではピクセルごとの並列処理が可能なため、GPUの得意とする処理です。

ST 2022-7 (redundancy)

M2S LibraryおよびM2svideosrc、m2svideosink、m2saudiosrc、m2saudiosinkはST2022-7 (redundancy) に対応しています。

ConnectX-6は送信・受信ともにST 2022-7にHW対応しています。一方で、ConnectX-5は送信のみST 2022-7にHW対応しています。
ConnectX-5でST 2022-7受信処理を実現するためにはRivermaxより上位のアプリケーションでソフトウェア処理を実装する必要があります。

M2S LibraryはConnectX-5/6両方での動作をサポートするため、ST 2022-7受信に関してはM2S Library内のソフトウェア処理で実現しています。一方で、ST 2022-7送信はConnextXのHW機能を利用しています。
データフローを以下に示します。（図表4）



図表4

GStreamerコマンドライン実例

それでは、GStreamerエレメントを組み合わせた実際のコマンドライン実例をいくつかご紹介いたします。

マクニカエレメントのパラメーター

初めに、m2svideosrc、m2svideosink、m2saudiosrc、m2saudiosinkに設定できるパラメーターを以下に示しておきます。（図表5)

パラメーター名	説明	備考
p-if-address	Interface IP Address of Primary	m2svideosrc、m2saudiosrcのみ
p-dst-address	Destination IP Address of Primary	
p-dst-port	Destination UDP port number of Primary	
p-src-address	Source IP Address of Primary	
s-if-address	Interface IP Address of Secondary	m2svideosrc、m2saudiosrcのみ
s-dst-address	Destination IP Address of Secondary	
s-dst-port	Destination UDP port number of Secondary	
s-src-address	Source IP Address of Secondary	
scan	scanning mode 0 : Progressive 1: TFF Interlaced 2: BFF Interlaced	m2svideosrc、m2svideosinkのみ
packet-time	Packet Time 0 : 1ms 1 : 125us	m2svideosrc、m2svideosinkのみ

図表5

▶ TX: Color Bar

GStreamerにはカラーバーを生成するvideotestsrcというエレメントがあります。このエレメントと組み合わせてカラーバーをST 2110で送信する例を示します。
1080i (2Kインターレース) での出力は以下の通りです。

```
$ gst-launch-1.0 videotestsrc
   ! video/x-raw,format=I420,width=1920,height=1080,framerate=30000/1001
   ! m2svideosink scan=1 p-dst-address="239.23.20.100" s-dst-address="239.23.21.100" p-src-address="192.168.1.23" s-src-address="192.168.2.23" p-dst-port=50020 s-dst-port=50020
```

次に2160p (4Kプログレッシブ) の出力です。上述のコマンドの解像度とフレームレートを変るだけ、では上手くいきませんでした。どうもvideotestsrcを3840x2160で動作させると処理が相当重くなるようです。そこで、videotestsrcからの出力はもっと小さな解像度とし、GStreamerのvideoscaleエレメントを使って4Kにアップコンバートする手法としました。具体的なコマンドは以下の通りです。

```
$ gst-launch-1.0 videotestsrc
  ! video/x-raw,format=I420,width=480,height=270,framerate=60000/1001
   ! videoscale ! video/x-raw,width=3840,height=2160
   ! m2svideosink scan=0 p-dst-address="239.23.20.100" s-dst-address="239.23.21.100" p-src-address="192.168.1.23" s-src-address="192.168.2.23" p-dst-port=50020 s-dst-port=50020
```

▶ TX: H.264 Decode

カラーバーだと動きがなくて面白くないので、ローカルストレージに保存されているH.264ファイルをリアルタイムにデコードしながらST 2110で送信してみましょう。2160p (4Kプログレッシブ) の出力例です。

```
$ gst-launch-1.0 filesrc location=BigBuckBunny.mp4
  ! qtdemux name=demux demux.video_0
   ! h264parse
   ! avdec_h264
   ! video/x-raw,format=I420
   ! videoscale ! video/x-raw,width=3840,height=2160
   ! videorate ! video/x-raw,framerate=60000/1001
   ! m2svideosink p-dst-address="239.23.20.100" s-dst-address="239.23.21.100" p-src-address="192.168.1.23" s-src-address="192.168.2.23" p-dst-port=50020 s-dst-port=50020
```

▶ モニター出力

次にST 2110の受信処理です。受信した映像データをデスクトップに描画してみましょう。デスクトップの描画にはximagesinkエレメントを使います。2160p (4Kプログレッシブ) の出力例です。

```
$ gst-launch-1.0 m2svideosrc scan=0 p-if-address="192.168.1.23" s-if-address="192.168.2.23" p-dst-address="239.160.20.101" s-dst-address="239.160.21.101" p-src-address="192.168.1.160" s-src-address="192.168.2.160" p-dst-port=50020 s-dst-port=50020
   ! video/x-raw,format=BGRx,width=3840,height=2160,framerate=60000/1001
   ! ximagesink display=:0
```

なお、上記コマンドは4K以上のモニター環境で実行しないと出力が画面からはみ出してしまいますので、モニターの解像度に合わせて出力をダウンコンバートする必要があります。さらに、モニターの解像度よりも小さい解像度で出力することにより、複数のST 2110ストリームをマルチビューアの様に表示することも可能です。

2160p (4Kプログレッシブ) を 480x270にダウンコンバートして出力する例は以下の通りです。

```
$ gst-launch-1.0 m2svideosrc scan=0 p-if-address="192.168.1.23" s-if-address="192.168.2.23" p-dst-address="239.160.20.100" s-dst-address="239.160.21.100" p-src-address="192.168.1.160" s-src-address="192.168.2.160" p-dst-port=50020 s-dst-port=50020
   ! video/x-raw,format=BGRx,width=3840,height=2160,framerate=60000/1001
   ! videoscale ! video/x-raw,width=480,height=270
   ! ximagesink display=:0
```

▶ RX: YouTube Live

最後に、ST 2110映像および音声を受信して、YouTube Liveでリアルタイム配信してみます。 YouTube Liveを利用するためには色々と準備が必要ですが、そのあたりの解説は他のサイトに譲るとして、ここでは具体的なコマンド例のみ示します。

```
$ gst-launch-1.0 m2svideosrc scan=1 p-if-address="192.168.1.23" s-if-address="192.168.2.23" p-dst-address="239.160.20.100" s-dst-address="239.160.21.100" p-src-address="192.168.1.160" s-src-address="192.168.2.160" p-dst-port=50020 s-dst-port=50020
   ! video/x-raw,format=I420,width=1920,height=1080,framerate=30000/1001
   ! videoconvert
   ! x264enc ! video/x-h264,stream-format=byte-stream
   ! h264parse
   ! queue
   ! flvmux name=mux streamable=true
   ! rtmpsink location=rtmp://a.rtmp.youtube.com/live2/xxxx-xxxx-xxxx-xxxx-xxxx
m2saudiosrc packet-time=0 p-if-address="192.168.1.23" s-if-address="192.168.2.23" p-dst-address="239.160.30.100" s-dst-address="239.160.31.100" p-src-address="192.168.1.160" s-src-address="192.168.2.160" p-dst-port=50030 s-dst-port=50030
  ! audio/x-raw,format=S24BE,rate=48000,channels=2,layout=interleaved
   ! audioconvert
   ! voaacenc bitrate=128000
   ! audio/mpeg
   ! aacparse
   ! audio/mpeg, mpegversion=4
   ! mux.
```

性能評価

2160p (4Kプログレッシブ) で同時動作できるストリーム数は以下の通りでした。（図表6）

解像度 / 周波数 / 帯域	2022-7	送信or受信	機能	最大ストリーム数
2160p / 59.94Hz / 12G	あり	送信	カラーバー	4ストリーム
			H.264デコード	4ストリーム
		受信	モニター出力	2ストリーム

図表6

ST 2110映像送信は、カラーバー、H.264デコードともに4ストリームまで同時動作が可能でした。ConnectX-5/6は100Gbps x 2ポートなので、イーサネットの帯域的には2022-7(Redundancy) の12Gストリームを8ストリーム送信できる計算になります。今回4ストリームが限界だったのは、実験結果からの推測になりますが、どうもPCIeの帯域がボトルネックとなっているようです。PCIeはGen3x16で動作しています。2022-7(Redundancy) のストリームを送信する際に、PCIe上は1ストリーム分のデータを流してConnectX側で2分配するのであればPCIeの帯域的にも8ストリーム流せる計算になるのですが、動作を見ていると、2022-7(Redundancy) 送信を行うためにはPCIe上も2ストリーム分の帯域を消費するようです。

一方で、ST 2110映像受信はモニター出力を2ストリームまで同時動作が可能でした。これはソフトウェア処理がボトルネックになっていると思われます。上述したように、受信に関しては2022-7(Redundancy)処理をM2S Library内のソフトウェア処理で行っています。そのため、送信よりもストリーム数が少ない結果になっているようです。この部分のパフォーマンス改善は今後の課題となります。また、これも上述したように、Connectx-6は2022-7(Redundancy)受信処理をHWで行う機能を備えています。この機能を利用することによりパフォーマンスの改善が期待できます。

おわりに

いかがでしたでしょうか。今回はGStreamerを取り上げましたが、画像/音声処理に関する魅力的なオープンソースは他にも多数存在します。ST 2110のベース部分をソフトウェアで実装することで、これらオープンソースが活用しやすくなります。皆さんもぜひ利用してみてください。

Co.Tomorrowing
MACNICA

株式会社マクニカ　クラビス カンパニー
本社
〒222-8561 横浜市港北区新横浜1-6-3 マクニカ第1ビル
TEL 045-470-9821 / FAX 045-470-9822

株式会社マクニカ　クラビス カンパニー
大阪オフィス
〒530-0005 大阪市北区中之島2-3-33 大阪三井物産ビル
TEL: 06-6227-6910　FAX：06-6227-6911

お問い合わせフォーム
https://www.macnica.co.jp/business/semiconductor/support/contact/

2021年 ©MACNICA, Inc.