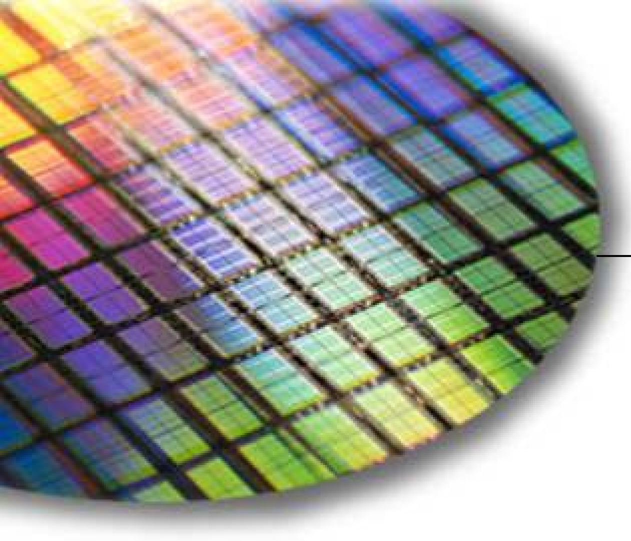


The World Leader in High Performance Signal Processing Solutions

Blackfin Device Drivers

Presented By:
David Lannigan

ANALOG DEVICES

About this Module

This module discusses the device driver model for the Blackfin family of processors.

It is recommended that users should have an understanding of the Blackfin architecture and is familiar with the Blackfin System Services software.

Module Outline

- **Overview**
  - **General Information**
    - Common conventions, terminology, return codes etc.
- **Device Driver API**
- **Buffers**
- **Dataflow Methods**
- **UART example (VisualDSP, December 2005 update)**

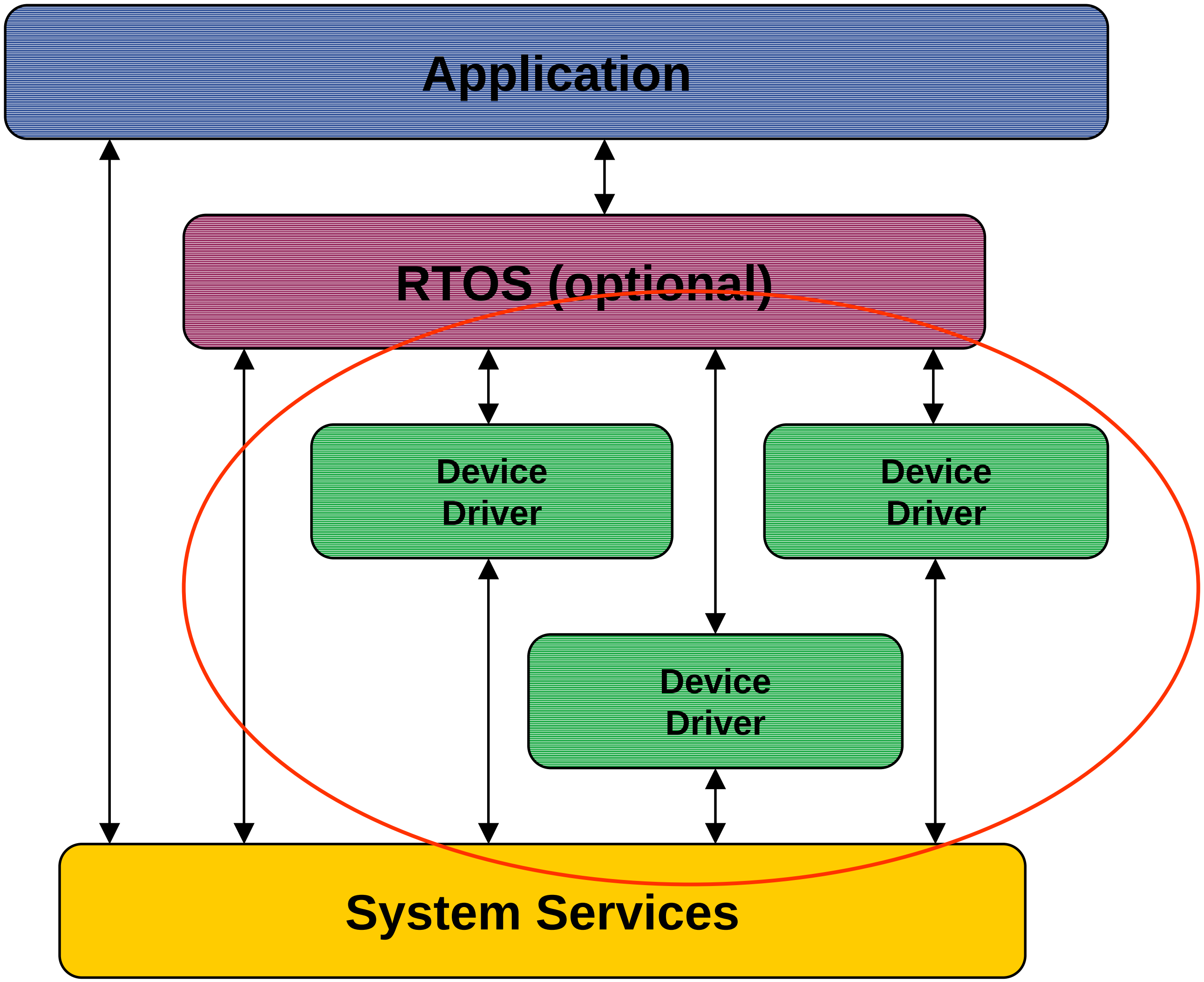
Device Driver Model

- **Standardized API for Blackfin processors**
  - **User interface is the same**
    - Regardless of driver
      - Allows buffers to be passed from driver to driver
    - Regardless of processor
      - Application using UART does not change from BF533 to BF537
  - **Developers only have to learn it once**
    - All drivers operate the same way
- **Extensible**
  - **Drivers can add their own commands (IOCTLs), events etc.**
- **Goal is to cover the vast majority**
  - **Exceptions will exist**

Leverages the System Services

- **Faster development**
  - **Stable software base**
    - Fewer variables
  - **Less re-invention**
    - Example: drivers do not need to include DMA code
- **Modular software**
  - **Better compatibility**
    - Resource control is managed by the system services
  - **Easier integration**
    - Multiple drivers working concurrently
- **Portability**
  - **Driver for the BF533 works on the BF561**

System Architecture (Drivers and Services)



Using Device Drivers in VisualDSP Projects

- **Application source file**
  - **#include <services/services.h>** **// system services**
  - **#include <drivers/adi_dev.h>** **// device manager**
  - **#include <drivers/xxx.h>** **// device driver's .h file**
- **Source file folder**
  - **For off-chip device drivers only**
    - Include the device driver's .c file in the list of source files
- **Linker files folder**
  - **Link with the proper libssl library**
    - Pulls in the system services
  - **Link with the proper libdrv library**
    - Pulls in on-chip device drivers

Off-chip drivers need to be explicitly included

On-chip drivers come in the library

- ***Use code elimination option in linker***
  - ***Significantly reduces code size***

Library Configurations

- **Debug version of on-chip driver library**
  - **Optimizations off**
  - **Symbolic information included**
  - **Can step into the sources**
  - **Lots of parameter checks**
  - **Should be the libraries users start with**
    - Examples/demos typically use the debug version
- **Release version on-chip driver library**
  - **Optimizations on**
  - **No symbolic information**
  - **Can't step into the sources**
  - **Few, if any, parameter checks**
  - **Should be the libraries users end with**
    - Users should release with the release version

Library Filenames (pg 1-12 in the manual)

libdrvaaa_bbbcc.dlb

aaa – processor variant

532 – BF531, BF532, BF533
534 – BF534, BF536, BF537
561 – BF561

cc – special conditions

d – debug version
y – workarounds for all silicon anomolies
dy – debug plus workarounds
blank – no debug, no workarounds

_bbb – operating environment

blank - standalone

i.e. libdrv532dy.dlb

- **ADSP-BF531, 532 or 533**
- **Debug plus workarounds**

Finding Device Drivers

- **Include files**
  - C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\include\drivers
- **Source files**
  - C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\lib\src\drivers
- **Libraries**
  - C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\lib
- **Examples**
  - C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\EZ-KITs\ADSP-BF533\Drivers
  - C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\EZ-KITs\ADSP-BF537\Drivers
  - C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\EZ-KITs\ADSP-BF561\Drivers
- **Documentation**
  - Device Driver and System Services User Manual
    - Blackfin Technical Library at www.analog.com
  - Device Driver and System Services User Manual Addendum (Sept 2005)
    - ftp://ftp.analog.com/pub/tools/patches/Blackfin/VDSP++4.0/

Memory

- No dynamic memory is used in the device drivers
- Device Manager needs memory to manage devices
  - Supplied by the application when initialized
  - Dictates how many simultaneously open drivers can be supported
- Physical drivers use static memory for their internal data
- No restrictions on memory placement
  - Code or data

Handles

- **Used in all device drivers** **(used in most services)**
  - **Device Handle ADI_DEV_HANDLE**
- **Unique identifier**
  - **Always typedef-ed to a void ***
  - **Is the address of something**
    - ADI_DEV_HANDLE – points to device specific data
- **Client Handle**
  - **Whatever the client wants it to be**
  - **No significance to the device drivers** **(or system services)**
  - **Can be anything that fits in a void ***
    - May point to something important for the client
    - May be a value of something
    - May be NULL

Return Codes

- Every (almost every) API function returns a code
  - Zero – universal success
  - Non-zero – some type of error or informative fact
- Each driver has its own set of return codes
  - All drivers return u32 for their return code
  - Pass back driver return codes OR system service return codes
- Resolving errors
  - Identify the driver or service that generated the error (services.h)
  - Find value in the .h file for that driver or service

Initialization Sequence

- **Initialize services in the following order**
  - **Omit any services not required**
    1. Interrupt Manager
    2. EBIU
    3. Power
    4. Port Control (BF534, BF536, BF537 only)
    5. Deferred Callback Service
    6. DMA Manager
    7. Flag Service
    8. Timer Service
- **After services are initialized, then initialize device drivers**
  - **adi_dev_Init()**

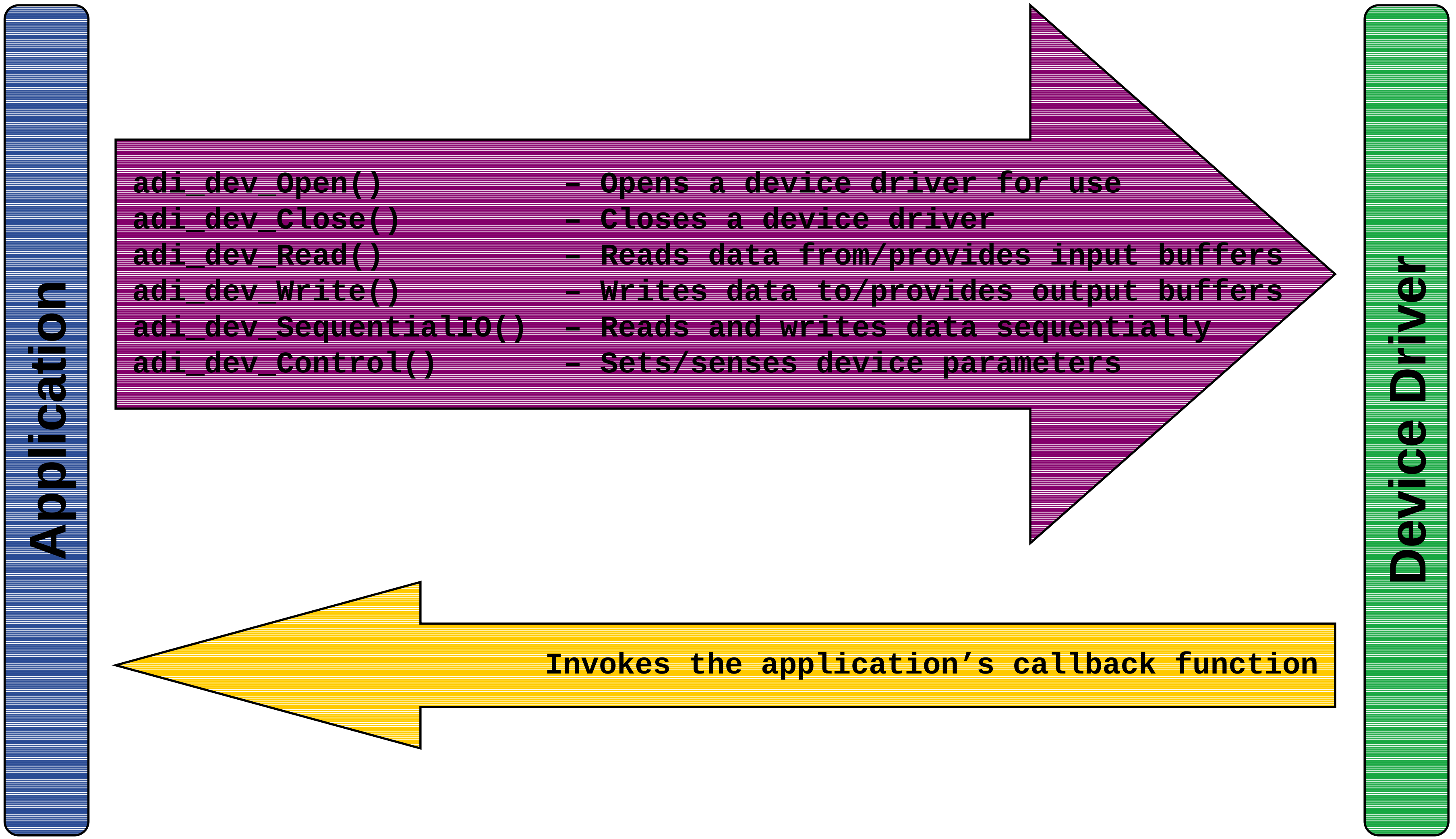
Termination Sequence

- **Termination is often not required in embedded systems**
  - **Do not call termination if not required**
    - Code elimination will optimize it out
- **Terminate device drivers before terminating any services**
  - **adi_dev_Terminate()**
- **Terminate services in the following order**
  - **Omit any services not required**
    1. Timer Service
    2. Flag Service
    3. DMA Manager
    4. Deferred Callback Service
    5. Port Control (BF534, BF536, BF537 only)
    6. Power
    7. EBIU
    8. Interrupt Manager

RTOS Considerations

- No device driver dependencies on RTOS
  - Hence, no VDK device driver library file
- All RTOS interactions isolated in the system services
  - Interrupt Manager
    - Critical regions
    - IMASK manipulations
    - Deferred callbacks
- Identical device driver API
  - VDK
  - Standalone

Device Driver API



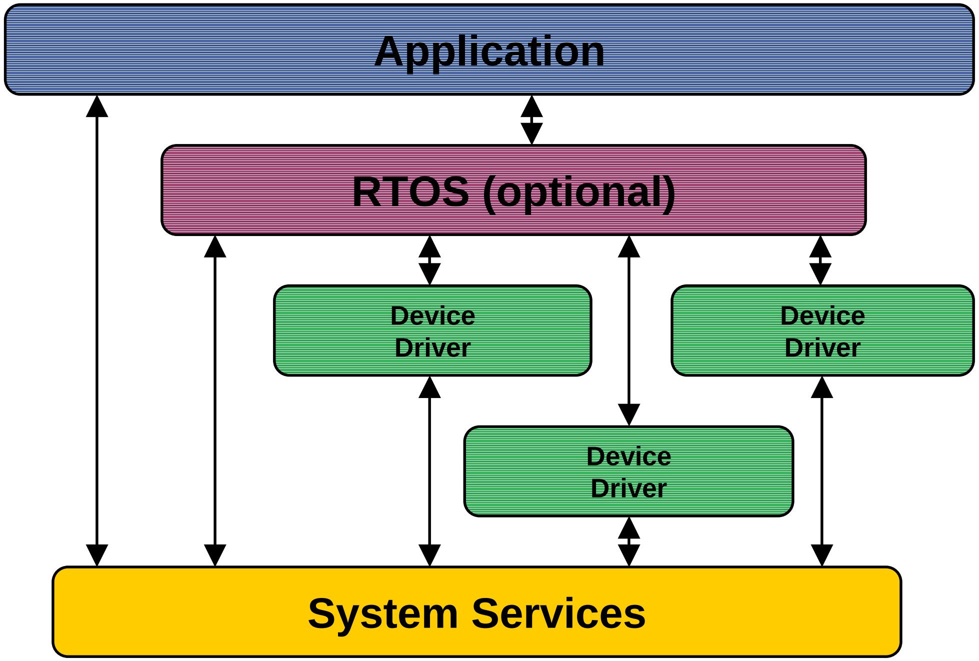
Device Driver API

- **Same API functions for all drivers (adi_dev.h)**
  - **UART, PPI, SPI, SPORT, TWI**
  - **ADC, DAC, video encoders, video decoders etc.**

- **Device driver extensions (adi_xxx.h)**
  - **Can add custom commands**
    - Passed in via adi_dev_Control()
  - **Can create new return codes**
    - Return values from each of the API functions
  - **Can create additional events**
    - Passed to the client via callback function

Device Drivers and System Services

- **Device drivers manage their own system services**
  - **Drivers call into system services as required**
    - e.g. PPI driver
      - Calls into DMA Manager
      - Calls into Interrupt Manager
      - Calls into Timer Control
      - Calls into DCB
  - **Application involvement**
    - Initialize services



Moving Data Through Device Drivers

- **Application provides buffers to device driver for processing**
  - **Buffers describe the data for the device driver to process**
    - Inbound buffers – Filled with data received from the device
    - Outbound buffers – Contain data to send out through the device
- **Buffer types**
  - **One dimensional** **ADI_DEV_1D_BUFFER**
    - Provided to driver via adi_dev_Read() or adi_dev_Write()
  - **Two dimensional** **ADI_DEV_2D_BUFFER**
    - Provided to driver via adi_dev_Read() or adi_dev_Write()
  - **Sequential (one dimensional)** **ADI_DEV_SEQ_1D_BUFFER**
    - Provided to driver via adi_dev_SequentialIO()
  - **Circular** **ADI_DEV_CIRCULAR_BUFFER**
    - Provided to driver via adi_dev_Read() or adi_dev_Write()

One Dimensional Buffer

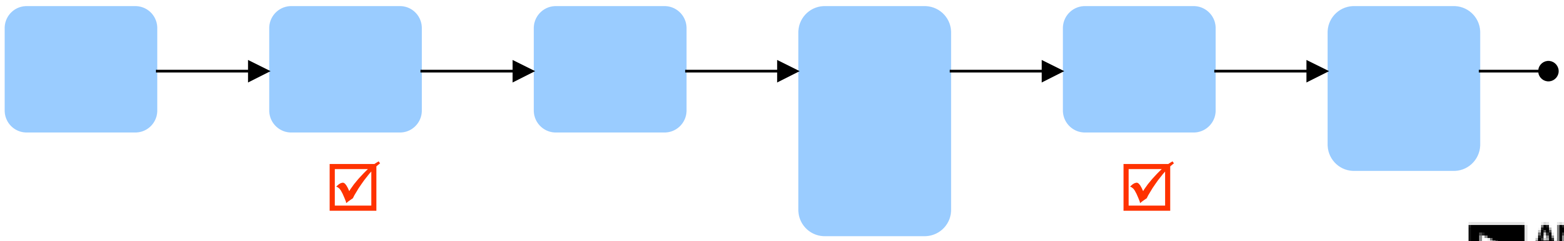
- **Pointer to the data**
  - Data can exist anywhere in memory
- **Element count**
  - Number of data elements
- **Element width**
  - Width, in bytes, of an element
- **Callback parameter**
  - NULL – no callback when the buffer is processed
  - Non-NULL – callback generated and this value passed to callback
- **Processed Flag** (filled in by device driver)
  - Set when the buffer is processed by the driver
- **Processed Count** (filled in by device driver)
  - Number of bytes processed in the buffer
- **pNext**
  - Pointer to the next buffer in the chain (NULL if the last/only buffer in chain)
- **pAdditionalInfo**
  - Pointer to any device specific information for the buffer
  - Not used by most devices

Dataflow Methods

- **Five dataflow methods**
  - **Chaining** — **1D and 2D buffers only**
  - **Chaining with loopback** — **1D and 2D buffers only**
  - **Sequential chaining** — **Sequential 1D buffers only**
  - **Sequential chaining with loopback** — **Sequential 1D buffers only**
  - **Circular** — **Circular buffers only**
- **Device drivers**
  - **Must support at least 1 dataflow method**
    - PPI
      - 1D, 2D and Circular
    - UART
      - 1D
    - TWI
      - Sequential 1D

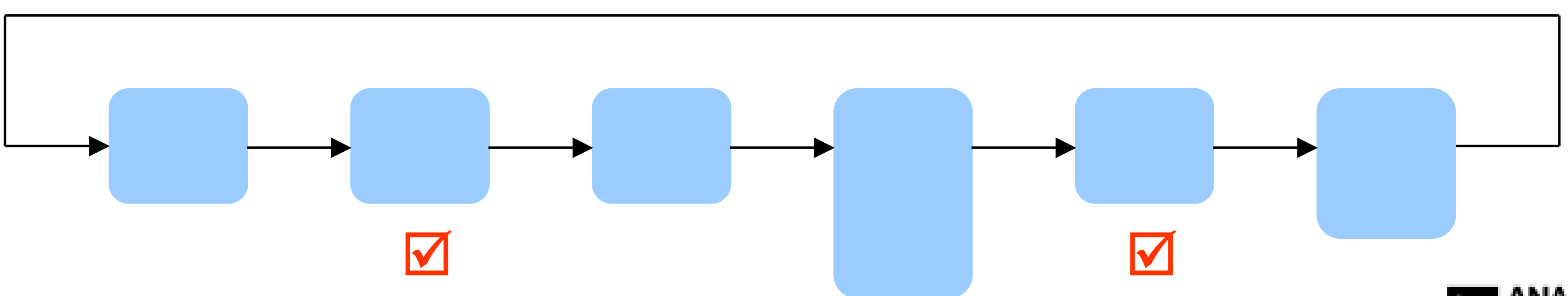
Chaining Method

- **Buffers are effectively "queued" to the device driver**
  - **Inbound buffers in one queue**
    - Buffers processed in the order they are received by adi_dev_Read()
  - **Outbound buffers in another queue**
    - Buffers processed in the order they are received by adi_dev_Write()
- **Buffers**
  - **Can be provided at any time**
  - **Submitted one at a time or in groups**
  - **Can point to data of different sizes**
- **Any, all or no buffers can be tagged to generate a callback**
- **Once processed, buffer is not used again unless resubmitted**



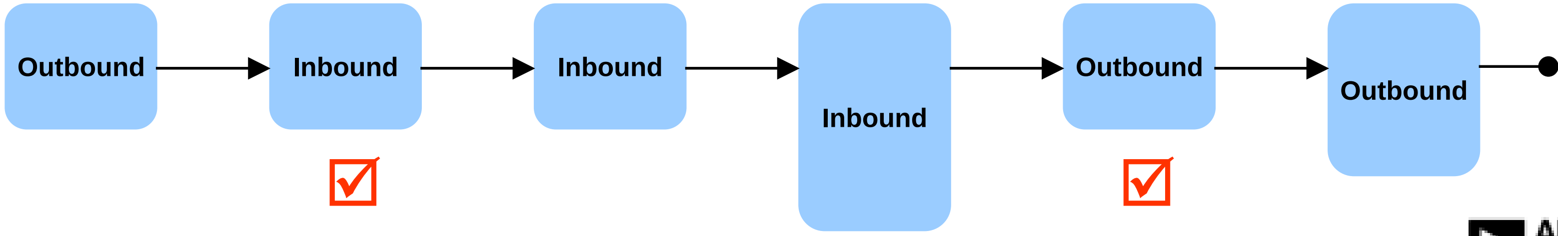
Chaining with Loopback Method

- **Identical to chaining method except:**
  - **Device driver automatically loops back to the first buffer after the last buffer in the chain is processed**
  - **Buffers can be provided only when dataflow is stopped**
- **Application does not need to re-supply buffers**
  - **Lower overhead**
  - **Device driver never "starves" for data**



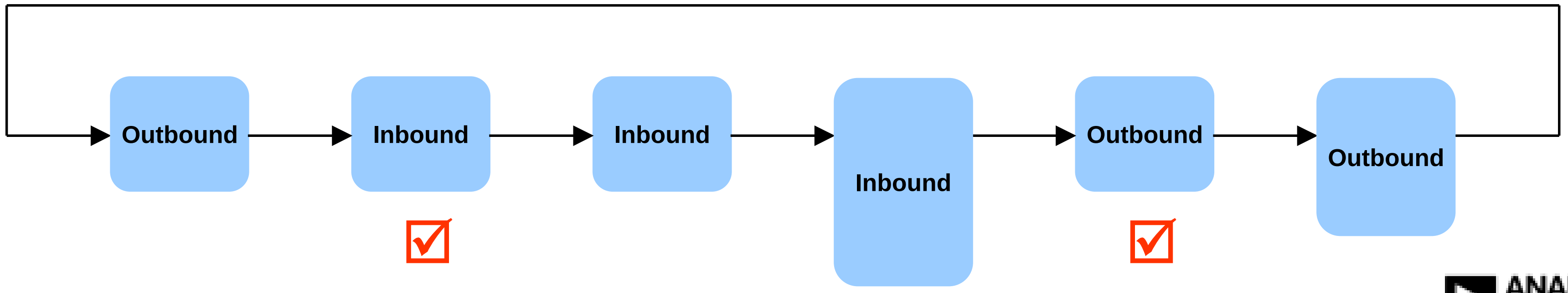
Sequential Chaining Method

- Buffers are effectively “queued” to the device driver
  - Field in buffer indicates direction (inbound or outbound)
  - Inbound and outbound buffers in one queue
    - Buffers processed in the order they are received by adi_dev_SequentialIO()
- Buffers
  - Can be provided at any time
  - Submitted one at a time or in groups
  - Can point to data of different sizes
- Any, all or no buffers can be tagged to generate a callback
- Once processed, buffer is not used again unless resubmitted



Sequential Chaining with Loopback Method

- **Identical to sequential method except:**
  - **Device driver automatically loops back to the first buffer after the last buffer in the chain is processed**
  - **Buffers can be provided only when dataflow is stopped**
- **Application does not need to re-supply buffers**
  - **Lower overhead**
  - **Device driver never "starves" for data**

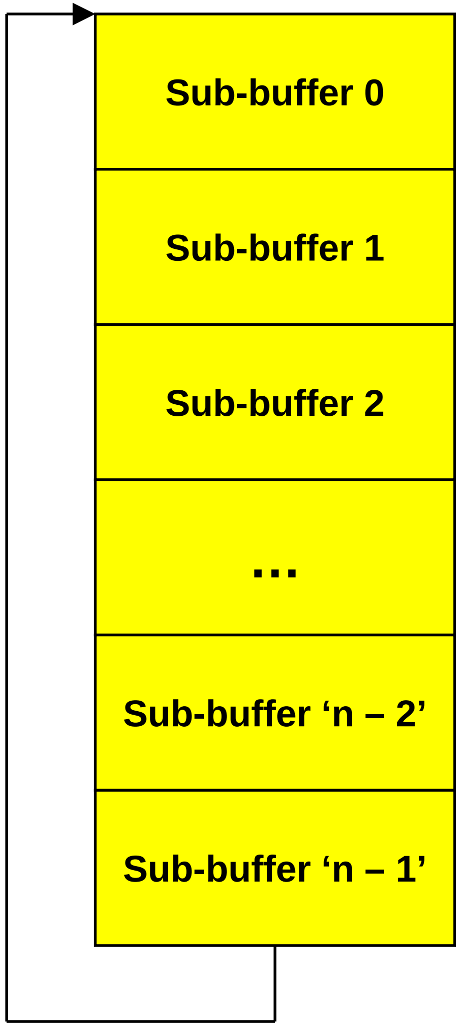


Streaming Command

- **Can be used with any chained dataflow method**
  - **Chaining, chaining w/loopback, sequential, sequential w/loopback**
- **Assertions to device driver**
  - **Application will insure the driver never runs out of buffers**
  - **If buffers with callbacks are used, system timing insures interrupts won't be lost**
- **Device drivers maximize throughput**
  - **Peripheral DMA supported devices use streaming DMA**
  - **Useful for audio and video**
    - Eliminates clicks, pops, glitches etc.

Circular Dataflow Method

- Single buffer provided to device
  - Define 'n' sub-buffers within the buffer
  - Buffer size limit of 64K bytes
- Automatic wrap-around
- Callbacks supported
  - None
  - Every sub-buffer completion
  - Whole buffer completion
- When supported by peripheral DMA
  - Leverages autobuffer capability

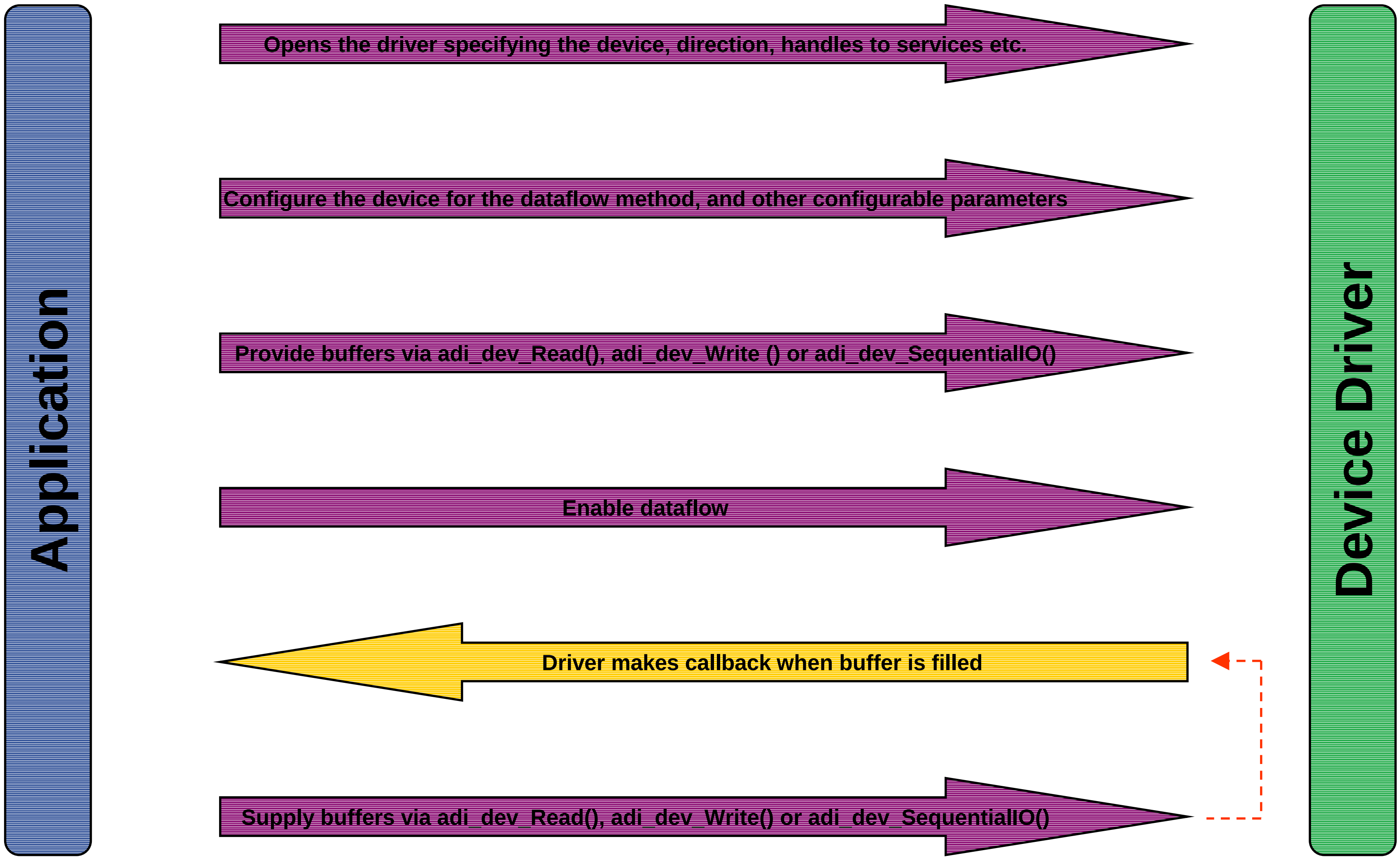


Deciding on a Dataflow Method

- **Circular**
  - **Data fits in a 64K byte contiguous block**
  - **Streaming type data**
  - **Audio frequently a candidate for circular dataflow**
- **Chained without loopback**
  - **Packet based data**
  - **Bursty data flow**
    - i.e. Ethernet, UART, USB etc.
- **Chained with loopback**
  - **Steady data flow**
  - **Audio, Video**
    - Use streaming to avoid clicks/pops/glitches
- **Sequential w/wout loopback**
  - **Half-duplex serial type devices**
    - TWI (I^2C compatible)

ANALOG DEVICES

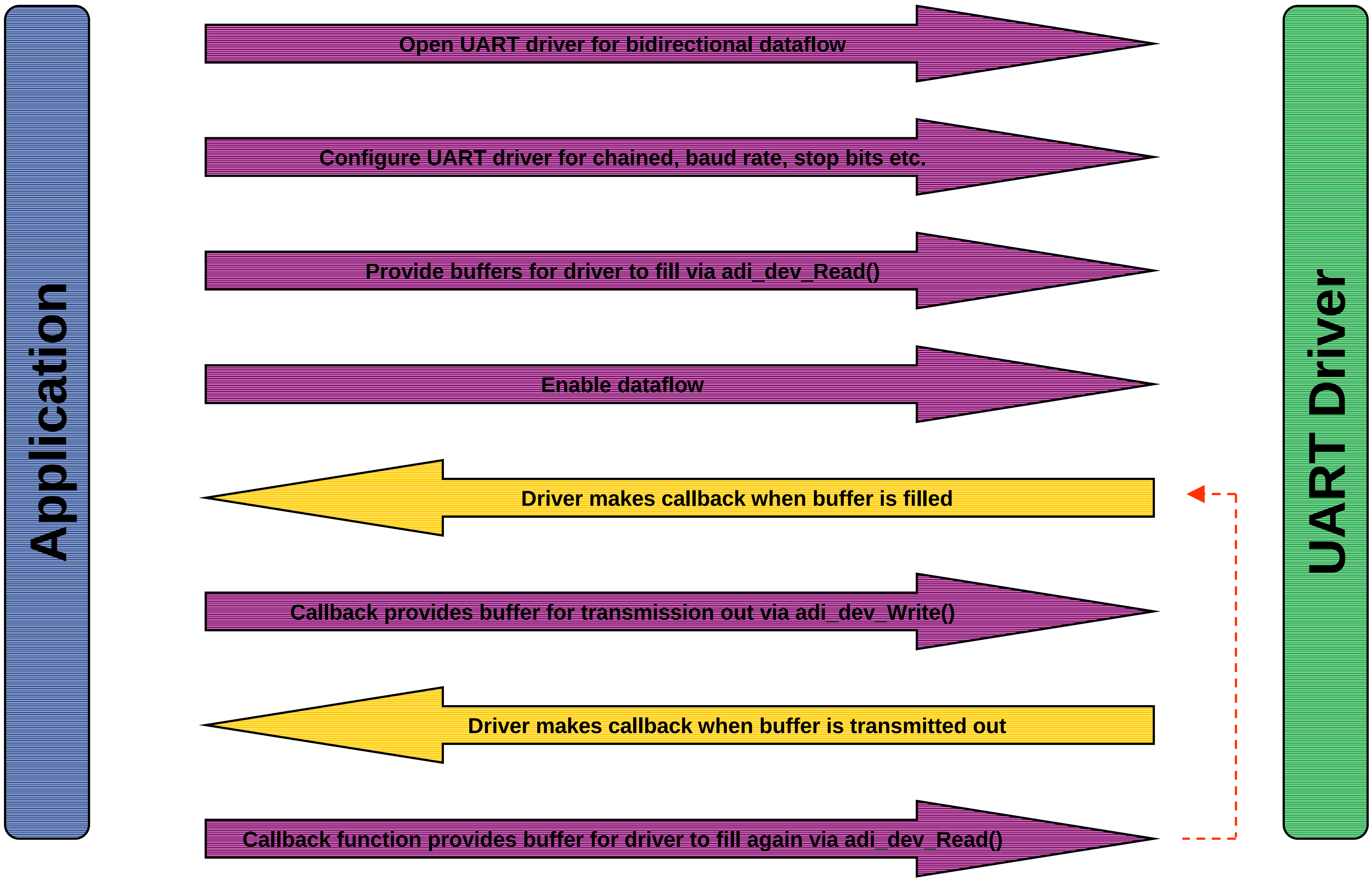
Typical Programming Sequence



UART Example

- **Echo program for UART device driver**
  - **Characters entered in terminal are received by the Blackfin program and echoed back to the terminal**
- **Connect serial cable from BF537 EZ-Kit to PC**
- **Start hyperterminal**
  - **57600 baud, 8 data bits, 1 stop bit, no parity**
- **Example demonstrates**
  - **Usage of the UART device driver**
  - **Chained dataflow method**
  - **Callbacks**

Programming Sequence for UART Example



ANALOG DEVICES

Conclusion

Device drivers provide:

- **Faster development**
  - **Stable software base for application development**
    - Fewer variables
  - **Less re-invention**
    - Do not need to create everything from scratch
- **Modular software**
  - **Better compatibility**
    - Resource control is managed by the system services
  - **Easier integration**
    - Multiple software components working concurrently
- **Portability**
  - **Code portable to other Blackfin processors**

Additional Information

- **Documentation**
  - **Device Drivers and System Services Manual for Blackfin Processors**
    - http://www.analog.com/processors/manuals
  - **Device Drivers and System Services Addendum (Sept 2005)**
    - ftp://ftp.analog.com/pub/tools/patches/Blackfin/VDSP++4.0/

- **For questions, click "Ask A Question" button or send an email to Processor.support@analog.com**