CEC Continuing Education Center

DesignNews

Techniques for Interfacing with Modern Sensors

DAY 3 : Sensor Driver Techniques Part 1

Sponsored by

Digi-Key ELECTRONICS

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

- Introduction to Modern Sensor Interfacing
- Designing Sensor Interfaces
- Sensor Driver Techniques Part 1
- Sensor Driver Techniques Part 2
- Leveraging C++ in Sensor Interfacing

General Sensor Driver Design Patterns

Technique	Complexity	Efficiency
Polling	Low	Low
Interrupt	Medium	Medium
DMA Driven	Medium	High

Which technique do you use the most for your drivers?

- Polling
- Interrupt
- DMA
- Other

Technique #1 – Polled Drivers

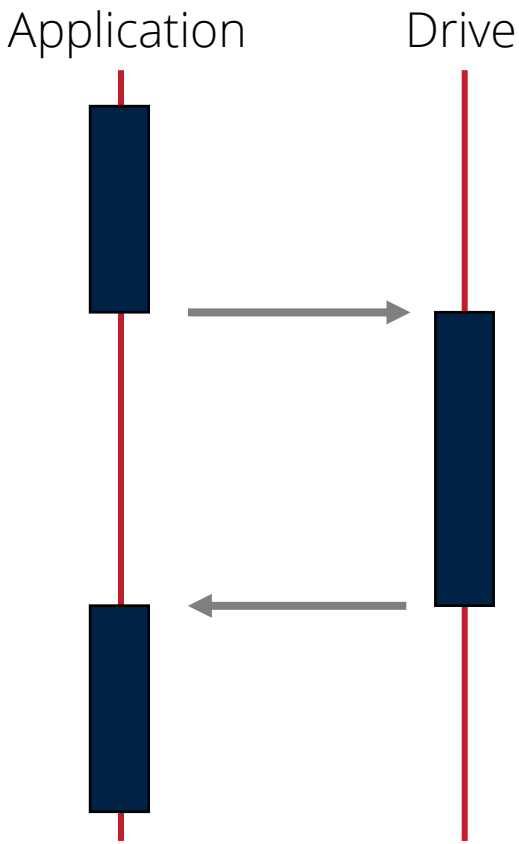
Advantages
- Simple to implement
- Fast to implement
- Low complexity

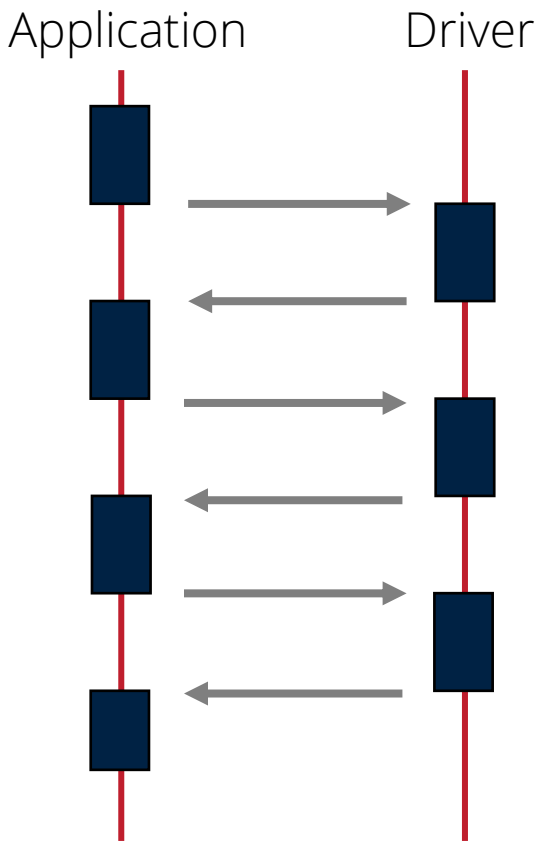
Disadvantages
- Inefficient
- May block code execution

Code Example:

```
uint16_t Adc_Sample(void)
{
   Adc_Start();

   while(ADC_COMPLETE_FLAG == FALSE);

   AdcResults = Adc_ReadAll();

   return AdcResults;
}
```

Technique #1 – Polled Drivers

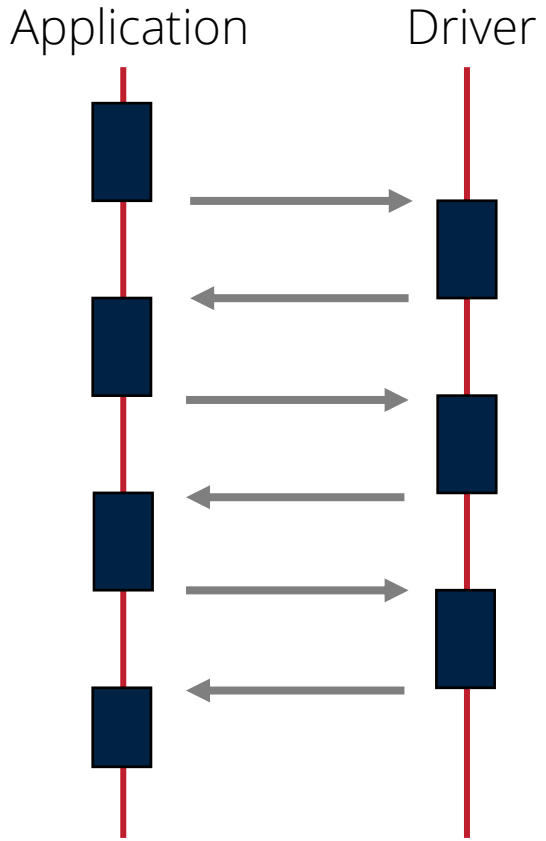




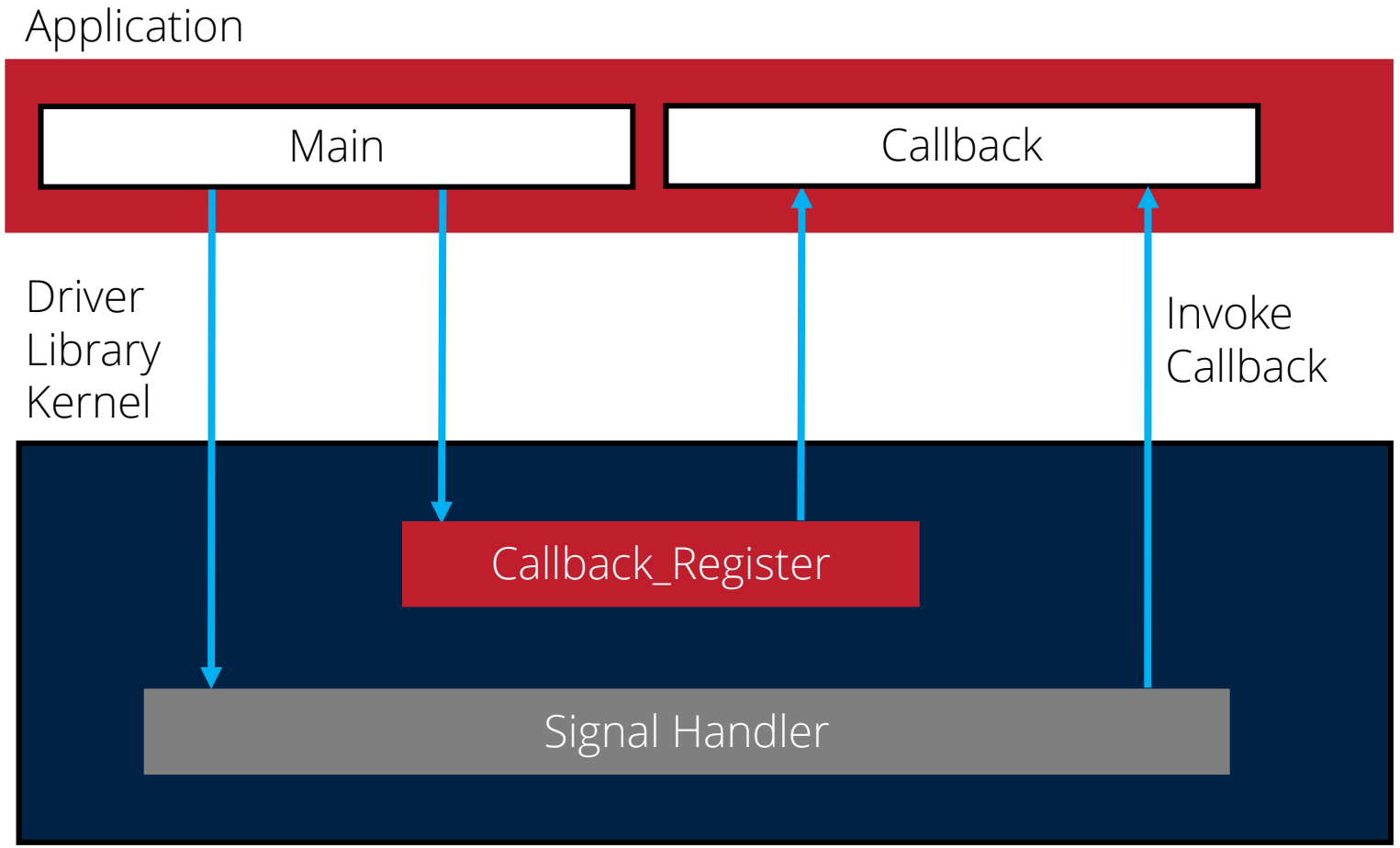
Technique #2 – Interrupt Driven Drivers

Advantages
- Efficient
- Easy to implement
- Can have run-time specified behavior

Disadvantages
- More complex to setup
- Potential context issues



Callbacks


Application
Main
Callback
Driver
Library
Kernel
Invoke
Callback
Callback_Register
Signal Handler

How often do you use callbacks in your software?

- 85 – 100% of the time
- 50 – 85% of the time
- 25 – 50% of the time
- Rarely, if ever

Interrupt Driven Driver Example - ADC

MPLAB® HARMONY

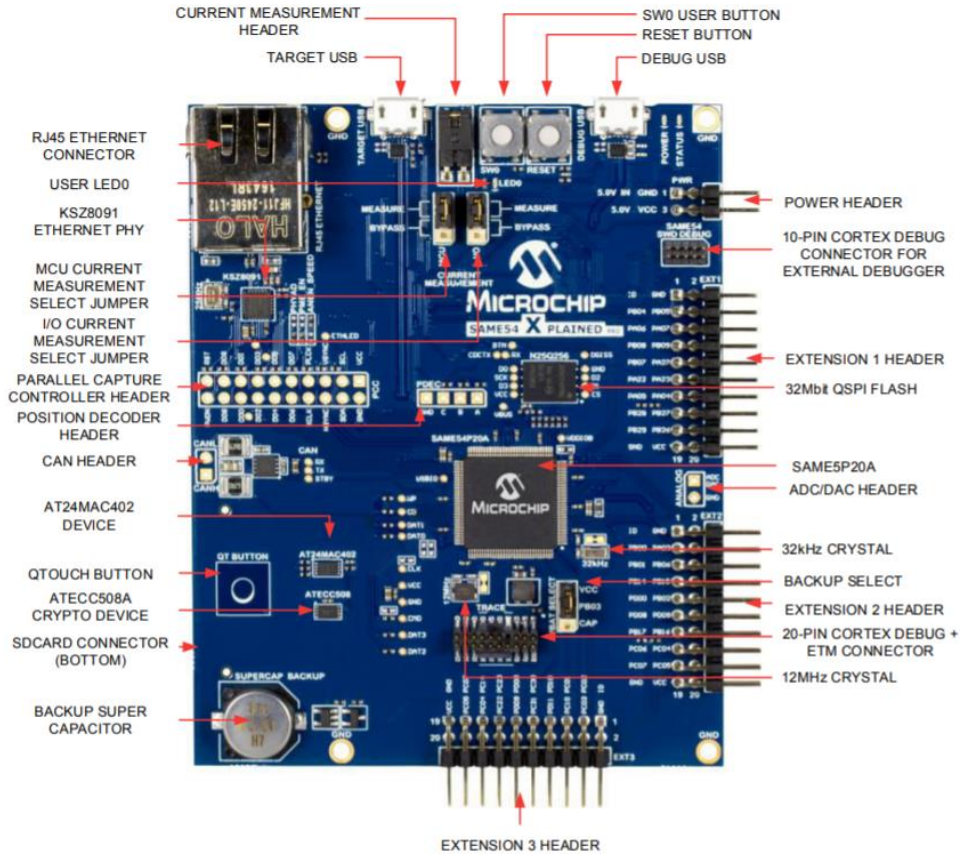


Image Source: https://microchipdeveloper.com/boards:sam-e54-xpro

Harmony ADC API's

```
void ADC0_Initialize( void );
void ADC0_Enable( void );
void ADC0_Disable( void );
void ADC0_ChannelSelect( ADC_POSINPUT positiveInput, ADC_NEGINPUT negativeInput );
void ADC0_ConversionStart( void );
uint16_t ADC0_ConversionResultGet( void );
void ADC0_ComparisonWindowSet(uint16_t low_threshold, uint16_t high_threshold);
void ADC0_WindowModeSet(ADC_WINMODE mode);
uint16_t ADC0_LastConversionResultGet( void );
void ADC0_InterruptsClear(ADC_STATUS interruptMask);
void ADC0_InterruptsEnable(ADC_STATUS interruptMask);
void ADC0_InterruptsDisable(ADC_STATUS interruptMask);
void ADC0_CallbackRegister( ADC_CALLBACK callback, uintptr_t context );
```

The ADC Interrupt

```
void ADC0_RESRDY_InterruptHandler( void )
{
    ADC_STATUS status;
    status = (ADC_STATUS) (ADC0_REGS->ADC_INTFLAG & ADC_INTFLAG_RESRDY_Msk);

    /* Clear interrupt flag */
    ADC0_REGS->ADC_INTFLAG = ADC_INTFLAG_RESRDY_Msk;

    if (ADC0_CallbackObject.callback != NULL)
    {
        ADC0_CallbackObject.callback(status, ADC0_CallbackObject.context);
    }
}
```

"Safety" check

The ADC Callback

```
void Adc_SampleCompleteCallback(ADC_STATUS status, uintptr_t context)
{
    AdcConversion_t * const AdcConversion = (AdcConversion_t * const )context;

    if(status == ADC_STATUS_RESRDY)
    {
        // Determine which conversion this is so we know where to store it.
        if(AdcConversion->Port == ADC_PORT_0)
        {
            Adc_Store_Port0(AdcConversion);
        }
        else
        {
            Adc_Store_Port1(AdcConversion);
        }
    }
    else
    {
        // Don't do anything.
    }
}
```

AdcConversion_t

```
/**
 * Defines a structure that maintains the adc conversion information for a
 * conversion that is progress.
 */
typedef struct
{
    AdcPort_t Port;         /**< Defines the adc port that will be converted */
    uint8_t Channel;        /**< Defines the channel on the port to convert */
    bool    IsComplete;     /**< Defines if the conversion is currently complete */
}AdcConversion_t;
```

Running the driver

```
AdcConversion_t AdcConversion0 = {ADC_PORT_0, ADC_INPUTCTRL_MUXPOS_AIN0, true};
AdcConversion_t AdcConversion1 = {ADC_PORT_1, ADC_INPUTCTRL_MUXPOS_AIN0, true};

// Register the ADC Callback that will save the sample and kick-off additional
// samples.
ADC0_CallbackRegister(Adc_SampleCompleteCallback, (uintptr_t)&AdcConversion0);
ADC1_CallbackRegister(Adc_SampleCompleteCallback, (uintptr_t)&AdcConversion1);

ADC0_Enable();
ADC1_Enable();
```

Running the driver

```
if(AdcConversion0.IsComplete == true && AdcConversion1.IsComplete == true)
{
    // Start sampling the next sensors
    AdcConversion0.IsComplete = false;
    AdcConversion0.Channel = 0;

    AdcConversion1.IsComplete = false;
    AdcConversion1.Channel = 0;

    ADC0_ChannelSelect( channel, ADC_INPUTCTRL_MUXNEG_GND );
    ADC1_ChannelSelect( channel, ADC_INPUTCTRL_MUXNEG_GND );

    ADC0_ConversionStart();
    ADC1_ConversionStart();
 }
```

Which of the following do you think best describes this technique?

- Simple and easy
- Complex but efficient
- Powerful and scalable
- I haven’t decided yet

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Techniques for Interfacing with Modern Sensors

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

informa markets