LECTURE 8: DIGITAL COMMUNICATION

REALTIDSSYSTEMER OG PROGRAMMERINGSSPROG

Sergiu Spataru
ssp@et.aau.dk

DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

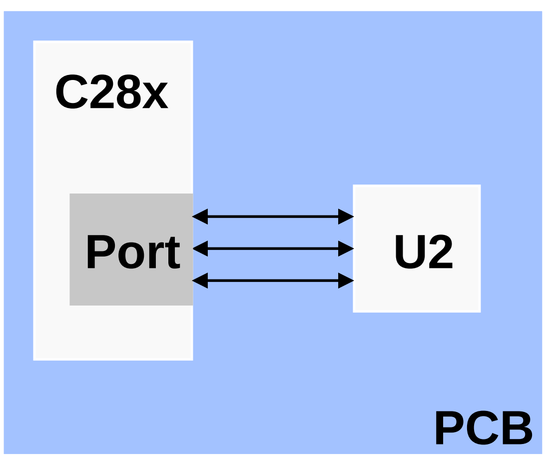
Agenda

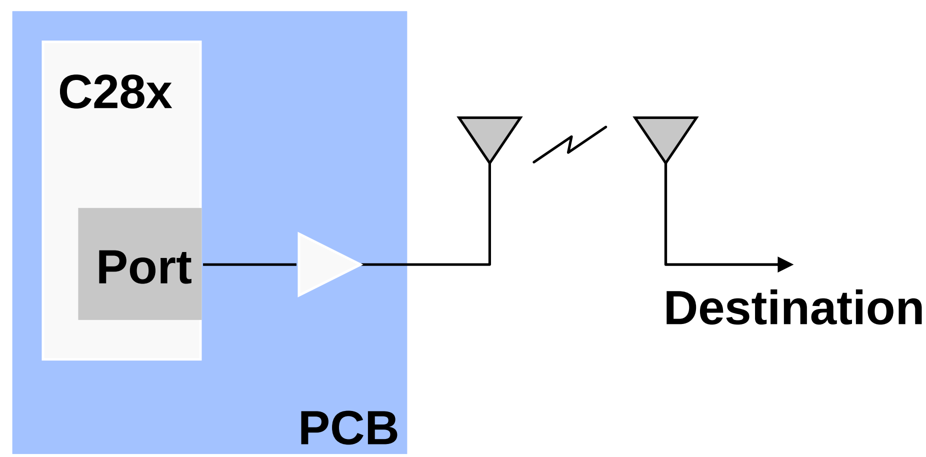
- **Asynchronous communication**
- SCI module of the F28069
- Implementing a communication protocol
- GUI Composer 2 introduction

Synchronous vs. Asynchronous

- Synchronous
  - Short distances (on-board)
  - High data rate
  - Explicit clock
  - Examples: Parallel, SPI, I2C

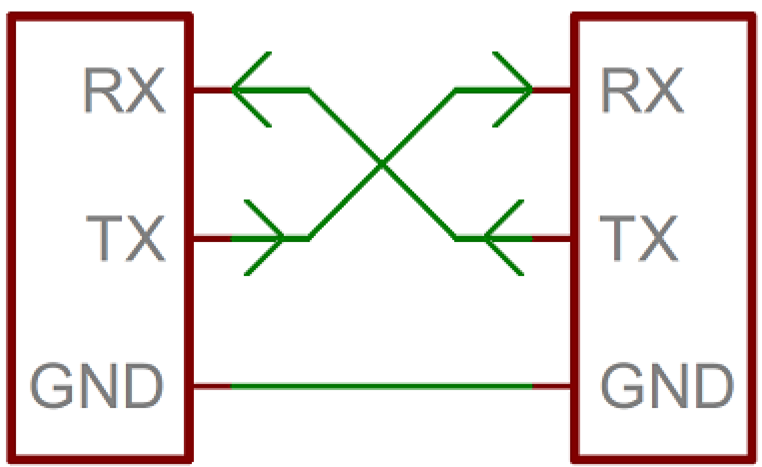
- Asynchronous
  - longer distances
  - Lower data rate (≈ 1/8 of SPI)
  - Implied clock (clk/data mixed)
  - Economical with reasonable performance
  - Examples: UART, CAN, USB, Ethernet





Asynchronous Serial Communication (UART)

- Implemented in a **universal asynchronous receiver-transmitter** (UART)
- Commonly used for communicating with GPS modules, Bluetooth, serial LCDs, instruments, PC
- Transmission is through 2 wires
  - TX for send data
  - RX for re



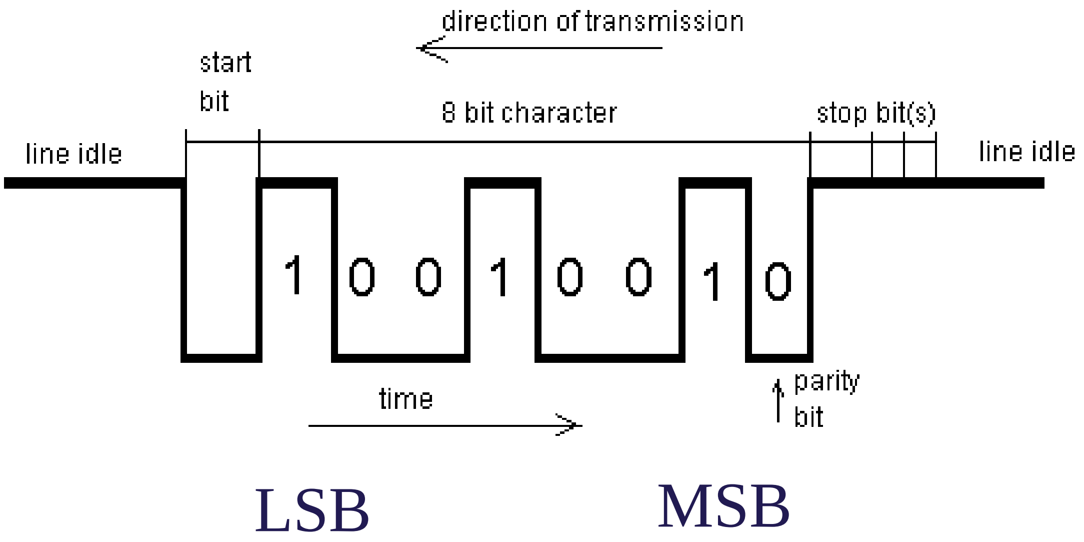
Asynchronous Serial Communication (UART)

- **Transmitted data (Data frame)** consists of:
  - 1 start bit
  - 5-9 data bits
  - 0 or 1 parity bitts
  - 1 or 2 stop bits(s)

- Transmitted at a preprogrammed **baud rate** (bits per second)
- Both the transmitter and receiver must be configured the same

Data Transfer

- Receiver resynchronizes again at a start of each new word (or character) received
- The send and receive of data is totally independent



DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

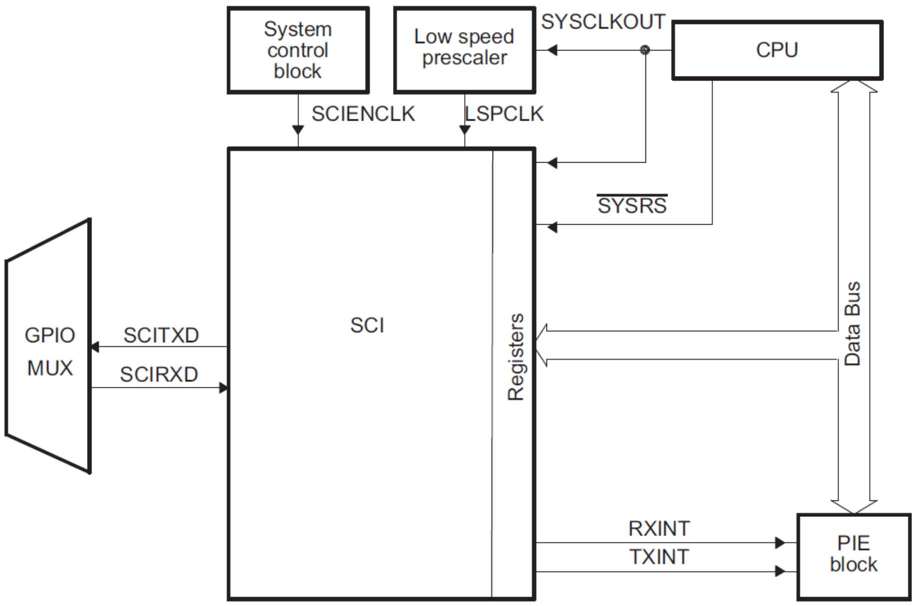
Agenda

- Asynchronous communication
- **SCI module of the F28069**
- Implementing a communication protocol
- GUI Composer 2 introduction

DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

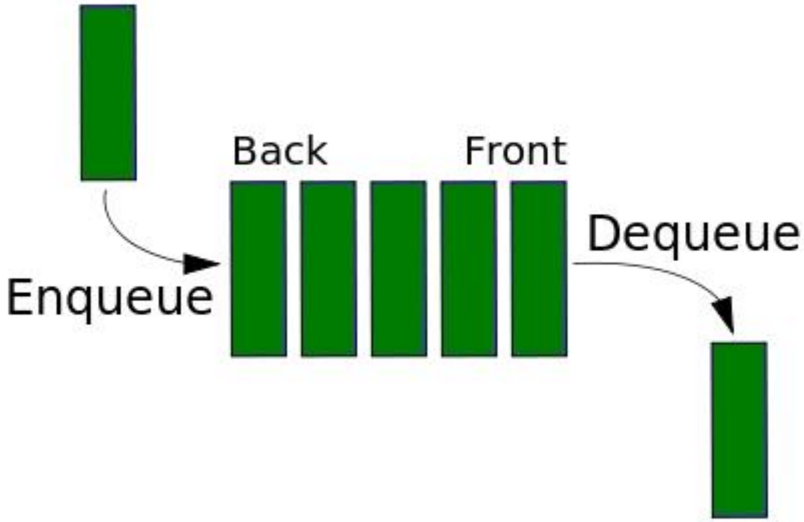
F28069 serial communications interface (SCI)/UART

- The F28069 has 2 SCI interfaces/UART Modules, denoted SCI-A, SCI-B
- Each has 2 GPIO pins (SCITXD**y**, SCIRXD**y, y=A or B**)
- Can generate an interrupt on data receive and transmit
- Has a 4 level deep FIFO (first in first out) buffer

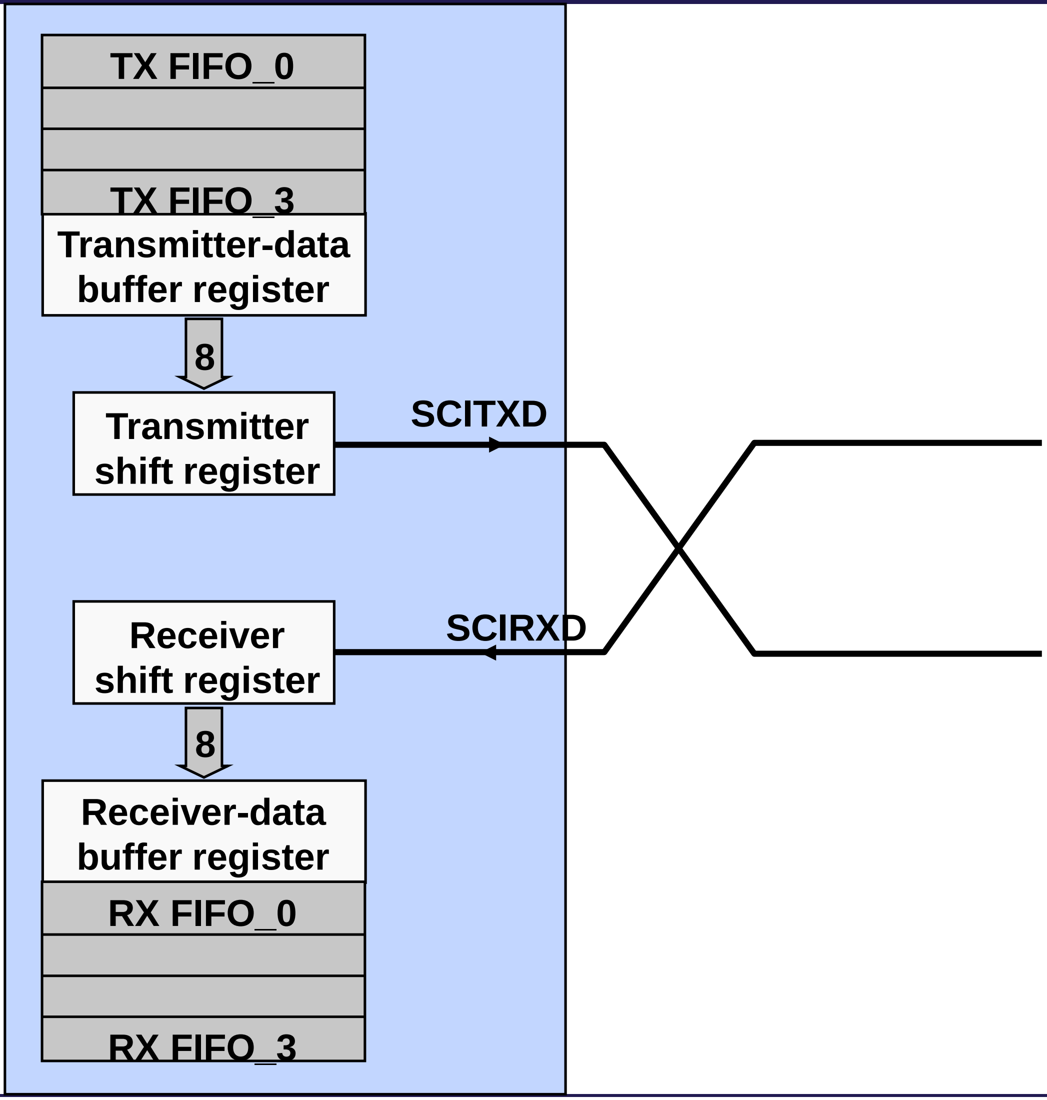


FIFO Buffers

- FIFO buffers are used to store data for digital transfer (received or transmit)



F28069 SCI FIFO buffers



Configuring the F28069 SCI

1. Configure the SCITXD and SCIRXD GPIO pins
    - (for SCIA these are GPIO28 and GPIO29)
2. Configure the data frame format
3. Configure the baud rate
4. Enable transmit and/or receive pins
5. Configure interrupt generation
6. (Optional) Configure interface for multiprocessor communication
7. Configure the transmit FIFO
8. Configure the receive FIFO
9. (Optional) Configure autobaudrate

Configure the data frame format

- Register definition files **SciaRegs** or **ScibRegs**

Start	LSB	2	3	4	5	6	7	MSB	Addr/ Data	Parity	Stop 1	Stop 2

This bit present only in Address-bit mode

Communications Control Register (Sci<u>x</u>Regs.SCICCR)

7	6	5	4	3	2	1	0
Stop Bits	Even/Odd Parity	Parity Enable	Loopback Enable	Addr/Idle Mode	SCI Char2	SCI Char1	SCI Char0

Stop Bits:
0 = 1 Stop bit
1 = 2 Stop bits

Even/Odd Parity:
0 = Odd
1 = Even

Parity Enable:
0 = Disabled
1 = Enabled

Loopback Enable:
0 = Disabled
1 = Enabled

Addr/Idle Mode:
0 = Idle-line mode
1 = Addr-bit mode

SCI Char2–Char0:
# of data bits = (binary + 1)
e.g. 110b gives 7 data bits

DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

Configure the data frame format

```
SciaRegs.SCICTL1.bit.SWRESET = 0; // Hold SCIA in reset mode
while configuring
```

While configuring hold SCI in reset

```
//// Configure data frame
SciaRegs.SCICCR.bit.STOPBITS = 0;// 1 Stop bit
SciaRegs.SCICCR.bit.LOOPBKENA  = 0;// Loop back disabled
SciaRegs.SCICCR.bit.PARITY = 0;// Odd parity
SciaRegs.SCICCR.bit.PARITYENA = 0;// Disable parity
SciaRegs.SCICCR.bit.SCICHAR = 111;// 8 data bits
```


DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

Configure the SCI Baud Rate

- SCI baud rate = $\dfrac{\text{LSPCLK}}{(\text{BRR} + 1) \times 8}$, BRR = 1 to 65535
- LSPCLK = 90MHz/4 = 22.5MHz (see Lecture 3 slide 49)

	LSPCLK Clock Frequency, 15 MHz		
Ideal Baud	BRR	Actual Baud	% Error
2400	780(30Ch)	2401	0.03
4800	390(186h)	4795	-0.10
9600	194(C2h)	9615	0.16
19200	97(61h)	19133	-0.35
38400	48(30h)	38265	-0.35

```
//High byte
SciaRegs.SCIHBAUD = 0x01;  // 9600 baud @LSPCLK = 22.5 MHz
//Low byte
SciaRegs.SCILBAUD = 0x24;
```


DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

Enable transmit and/or receive pins

Control Register 1 (SCICTL1)

7	6	5	4	3	2	1	0
reserved	RX ERR INT ENA	SW RESET	reserved	TXWAKE	SLEEP	TXENA	RXENA

0 = receiver disabled
1 = receiver enabled

0 = transmitter disabled
1 = transmitter enabled

0 = sleep mode disabled
1 = sleep mode enabled

Transmitter wakeup method select
1 = wakeup mode depends on SCICCR.3
0 = no wakeup mode

Write 0 = Reset SCI
Write 1 = release from Reset

0 = Receive Error Interrupt disabled
1 = Receive Error Interrupt enabled

Configure interrupt generation

SCICTL2

15 - 8	7	6	5 - 2	1	0
reserved	TXRDY	TX EMPTY	reserved	RX/BK INT ENA	TX INT ENA

SCI RX/BK INT ENA

0 = Disable RXRDY/BRKDT interrupt
1 = Enable RXRDY/BRKDT interrupt

SCI TX EMPTY

0 = TXBUF or shift register are loaded with data
1 = Transmit buffer and shift register both empty

SCI TX READY

0 = SCITXBUF is full
1 = SCITXBUF is empty

SCI TX INT ENA

0 = Disable TXRDY interrupt
1 = Enable TXRDY interrupt

Programming example

```
//Configure communication interface
SciaRegs.SCICTL1.bit.RXENA = 1; //Enable Rx (receive) pin
SciaRegs.SCICTL1.bit.TXENA = 1; //Enable Tx (transmit) pin
SciaRegs.SCICTL1.bit.RXERRINTENA = 0; //Disable receive error interrupt

SciaRegs.SCICTL1.bit.SLEEP = 0; //Disable sleep mode
SciaRegs.SCICTL1.bit.TXWAKE = 0; //No wakeup mode

SciaRegs.SCICTL2.bit.TXINTENA  = 1;//Enable transmit interrupt
SciaRegs.SCICTL2.bit.RXBKINTENA  = 1;//Enable receive interrupt

SciaRegs.SCIHBAUD = 0x01;
SciaRegs.SCILBAUD = 0x24;
```

Programming example - Interrupts

```
EALLOW;
PieVectTable.SCITXINTA = &SCIA_TX_isr;
PieVectTable.SCIRXINTA = &SCIA_RX_isr;
EDIS;

PieCtrlRegs.PIEIER9.bit.INTx2 = 1;// Enable SCIA TX interrupt
PieCtrlRegs.PIEIER9.bit.INTx1 = 1;// Enable SCIA RX interrupt

IER = M_INT9;
```

Configure the transmit FIFO

SCIFFTX

SCI Reset
0 = reset
1 = enable operation

SCI FIFO Enhancements
0 = disable
1 = enable

TX FIFO Reset
0 = reset (pointer to 0)
1 = enable operation

TX FIFO Status (read-only)

00000	TX FIFO empty
00001	TX FIFO has 1 word
00010	TX FIFO has 2 words
00011	TX FIFO has 3 words
00100	TX FIFO has 4 words

15	14	13	12	11	10	9	8
SCIRST	SCIFFENA	TXFIFO RESET	TXFFST4	TXFFST3	TXFFST2	TXFFST1	TXFFST0

7	6	5	4	3	2	1	0
TXFFINT	TXFFINT CLR	TXFFIENA	TXFFIL4	TXFFIL3	TXFFIL2	TXFFIL1	TXFFIL0

TX FIFO Interrupt Flag (read-only)
0 = not occurred
1 = occurred

TX FIFO Interrupt Flag Clear
0 = no effect
1 = clear

TX FIFO Interrupt (on match) Enable
0 = disable
1 = enable

TX FIFO Interrupt Level
Interrupt when TXFFST4-0 and TXFFIL4-0 match

Configure the transmit FIFO

```
SciaRegs.SCIFFTX.bit.SCIRST = 1; //Relinquish FIFO unit from reset
SciaRegs.SCIFFTX.bit.SCIFFENA = 1; //Enable FIFO- Enhancements
SciaRegs.SCIFFTX.bit.TXFIFOXRESET = 1;//Enable TX FIFO Operation


SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;//Clear TXFFINT-Flag
SciaRegs.SCIFFTX.bit.TXFFIENA = 1;//Enable TX FIFO match
SciaRegs.SCIFFTX.bit.TXFFIL = 0;//Set FIFO interrupt level to interrupt, if FIFO is empty (0)
```

Configure the Receive FIFO

SCIFFRX

RX FIFO Overflow Flag (read-only)
0 = no overflow
1 = overflow

RX FIFO Overflow Flag Clear
0 = no effect
1 = clear

RX FIFO Reset
0 = reset (pointer to 0)
1 = enable operation

RX FIFO Status (read-only)
00000 RX FIFO empty
00001 RX FIFO has 1 word
00010 RX FIFO has 2 words
00011 RX FIFO has 3 words
00100 RX FIFO has 4 words

15	14	13	12	11	10	9	8
RXFF-OVF	RXFF-OVF CLR	RXFIFO RESET	RXFFST4	RXFFST3	RXFFST2	RXFFST1	RXFFST0

7	6	5	4	3	2	1	0
RXFFINT	RXFFINT CLR	RXFFIEN	RXFFIL4	RXFFIL3	RXFFIL2	RXFFIL1	RXFFIL0

RX FIFO Interrupt Flag (read-only)
0 = not occurred
1 = occurred

RX FIFO Interrupt Flag Clear
0 = no effect
1 = clear

RX FIFO Interrupt (on match) Enable
0 = disable
1 = enable

RX FIFO Interrupt Level
Interrupt when RXFFST4-0 and RXFFIL4-0 match

Configure the Receive FIFO

```
SciaRegs.SCIFFRX.bit.RXFIFORESET = 1; //Enable RX FIFO Operation
SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1;//Clear TXFFINT-Flag
SciaRegs.SCIFFRX.bit.RXFFOVRCLR = 1; //Clear overflow flag
SciaRegs.SCIFFRX.bit.RXFFIENA = 1;//Enable RX FIFO match

SciaRegs.SCIFFRX.bit.RXFFIL = 2;//Set FIFO interrupt level to
interrupt, if FIFO received 2 chars

SciaRegs.SCICTL1.bit.SWRESET = 1; // Release SCIA from reset
mode while configuring
```

After configuration is doe
release SCI from reset

Agenda

- Asynchronous communication
- SCI module of the F28069
- **Implementing a communication protocol**
- GUI Composer 2 introduction

Receiving and transmitting integer data (2 bytes)

```c
struct INT16_BYTES {
    Uint16 BYTE0:8;  // 7:0
    Uint16 BYTE1:8;  // 15:8
};



union INT16_DATA_PACKAGE {
    int16 number;
    struct INT16_BYTES bytes;
};

union INT16_DATA_PACKAGE receivedData, dataToSend;
```

Reading received data integer data (2 bytes)

```
interrupt void SCIA_RX_isr(void)
{
receivedData.bytes.BYTE0 = SciaRegs.SCIRXBUF.bit.RXDT;
receivedData.bytes.BYTE1 = SciaRegs.SCIRXBUF.bit.RXDT;

//Load data into temporary buffer
dataToSend.number = receivedData.number * 2;

//Send signal to SCI to send the data
SciaRegs.SCIFFTX.bit.TXFIFOXRESET = 1;
SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;

SciaRegs.SCIFFRX.bit.RXFIFORESET = 0; // reset pointer
SciaRegs.SCIFFRX.bit.RXFIFORESET = 1; // enable op.
SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1; // reset RX int
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

SciaRegs.SCIFFRX .bit.RXFFIL = 2

Interrupt generated when at least 2 bytes have been received and are in FIFO

Transmitting data integer data (2 bytes)

```
interrupt void SCIA_TX_isr(void)
{
   SciaRegs.SCITXBUF = dataToSend.bytes.BYTE0;
   SciaRegs.SCITXBUF = dataToSend.bytes.BYTE1;

   PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

SciaRegs.SCIFFTX.bit.TXFFIL = 0
TX interrupt generated when FIFO buffer is empty
You can load up to 4 bytes in the FIFO buffer at once

DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

Receiving and transmitting integer data (2 bytes)

```
struct INT16_BYTES {
    Uint16 BYTE0:8;  // 7:0
    Uint16 BYTE1:8;  // 15:8
};



union INT16_DATA_PACKAGE {
    int16 number;
    struct INT16_BYTES bytes;
};

union INT16_DATA_PACKAGE receivedData, dataToSend;
```

Receiving and transmitting float data (4 bytes)

```
struct FLOAT32_BYTES {
    Uint32 BYTE0:8;  // 7:0
    Uint32 BYTE1:8;  // 15:8
    Uint32 BYTE2:8;  // 23:16
    Uint32 BYTE3:8;  // 31:24
};

typedef union {
    float number;
    struct FLOAT32_BYTES bytes;
}FLOAT32_DATA_PACKAGE ;

FLOAT32_DATA_PACKAGE dataToSend, receivedData;
```

Reading received data float data (4 bytes)

```
interrupt void SCIA_RX_isr(void)
{
receivedData.bytes.BYTE0 = SciaRegs.SCIRXBUF.bit.RXDT;
receivedData.bytes.BYTE1 = SciaRegs.SCIRXBUF.bit.RXDT;
receivedData.bytes.BYTE2 = SciaRegs.SCIRXBUF.bit.RXDT;
receivedData.bytes.BYTE3 = SciaRegs.SCIRXBUF.bit.RXDT;

//Load data into temporary buffer
dataToSend.number = receivedData.number * 2;

//Send signal to SCI to send the data
SciaRegs.SCIFFTX.bit.TXFIFOXRESET = 1;
SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;

SciaRegs.SCIFFRX.bit.RXFIFORESET = 0; // reset pointer
SciaRegs.SCIFFRX.bit.RXFIFORESET = 1; // enable op
SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1; // reset RX int
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

SciaRegs.SCIFFRX .bit.RXFFIL = 4

Interrupt generated when at 4 bytes have been received and are in FIFO

DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

Transmitting data float data (4 bytes)

```
interrupt void SCIA_TX_isr(void)
{
   SciaRegs.SCITXBUF = dataToSend.bytes.BYTE0;
   SciaRegs.SCITXBUF = dataToSend.bytes.BYTE1;
   SciaRegs.SCITXBUF = dataToSend.bytes.BYTE2;
   SciaRegs.SCITXBUF = dataToSend.bytes.BYTE3;

   PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

SciaRegs.SCIFFTX.bit.TXFFIL = 0
TX interrupt generated when FIFO buffer is empty
You can load up to 4 bytes in the FIFO buffer at once

Receiving and transmitting data with multiple bytes

```
Uint16 receivedBuffer[5];
Uint16 transmitBuffer[5];
Uint16 numReceivedWords = 0, numToSendWords;

interrupt void SCIA_TX_isr(void)
{
    int i, lastWordIndex;

    lastWordIndex = numToSendWords - 4;
    if (lastWordIndex < 0)
        lastWordIndex = 0;

    for(i = numToSendWords-1; i >= lastWordIndex; i--)
    {
        SciaRegs.SCITXBUF = transmitBuffer[i];     // Send data
        numToSendWords--;
    }

    if (numToSendWords > 0)
    {
        SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;
    }

   PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

Receiving and transmitting data with multiple bytes

```
interrupt void SCIA_RX_isr(void)
{
    int i;
    Uint16 command, response;
    for(i=0; i < SciaRegs.SCIFFRX.bit.RXFFST; i++)
    {
        receivedBuffer[numReceivedWords] = SciaRegs.SCIRXBUF.bit.RXDT;
        numReceivedWords++;
    }

    if (numReceivedWords == 5)
    {
        numReceivedWords = 0;
        receivedData.bytes.BYTE0 = receivedBuffer[1];
        receivedData.bytes.BYTE1 = receivedBuffer[2];
        receivedData.bytes.BYTE2 = receivedBuffer[3];
        receivedData.bytes.BYTE3 = receivedBuffer[4];
        command = receivedBuffer[0];

        response = CommunicationProtocol(command, &receivedData, &dataToSend);

        transmitBuffer[4] = response;
        transmitBuffer[3] = dataToSend.bytes.BYTE0;
        transmitBuffer[2] = dataToSend.bytes.BYTE1;
        transmitBuffer[1] = dataToSend.bytes.BYTE2;
        transmitBuffer[0] = dataToSend.bytes.BYTE3;
        numToSendWords = 5;

        //Send signal to SCI to send the data
        SciaRegs.SCIFFTX.bit.TXFFINTCLR = 1;

    }
SciaRegs.SCIFFRX.bit.RXFFINTCLR = 1; // reset RX int

PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}
```

Receiving and transmitting data with multiple bytes

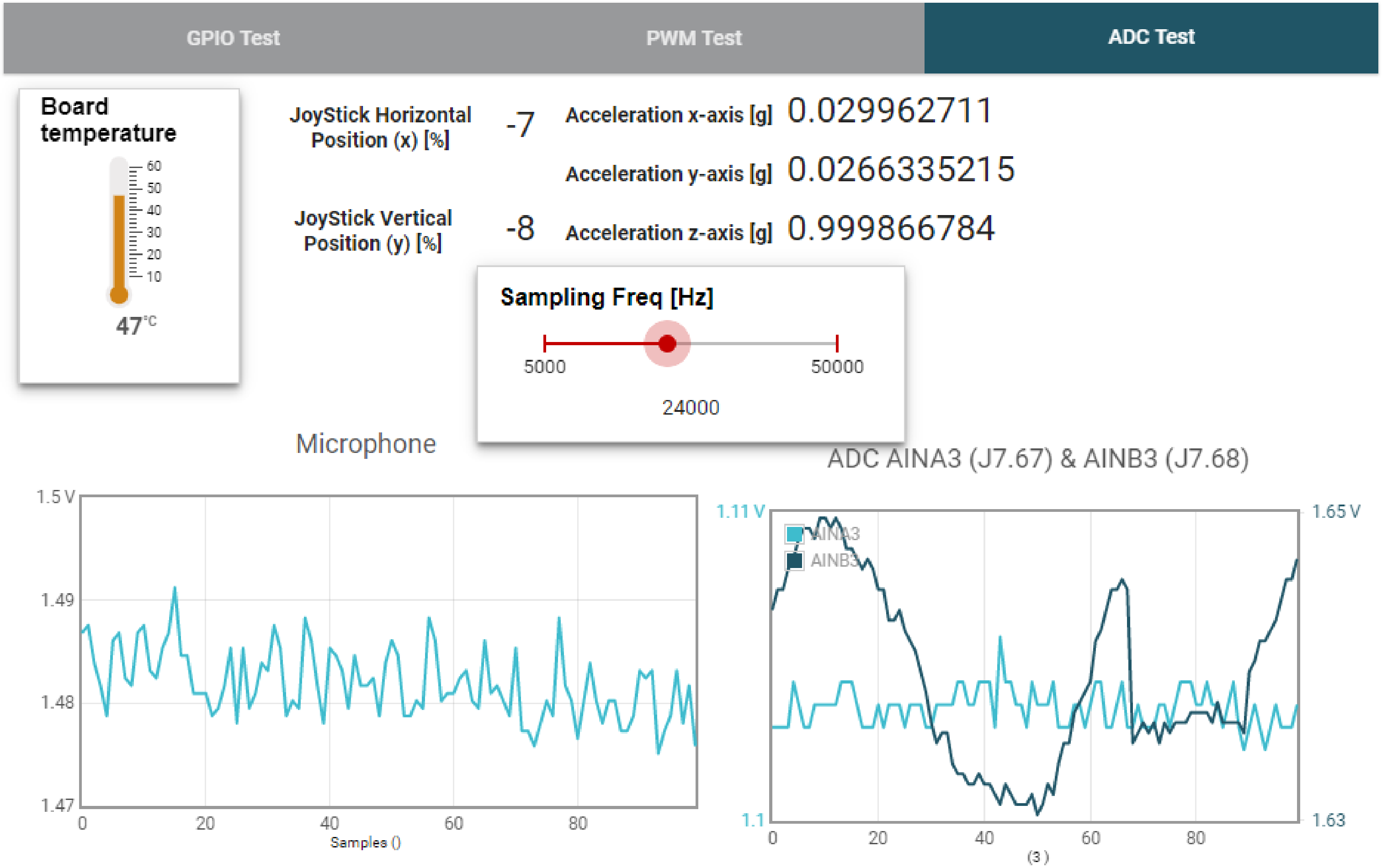
```
Uint16 CommunicationProtocol(Uint16 command, FLOAT32_DATA_PACKAGE* receivedPackage, FLOAT32_DATA_PACKAGE*
toSendPackage)
{
    Uint16 response;

    switch (command)
    {
        case 1:
        {
            signal1 = receivedPackage->number;
            toSendPackage->number =  signal1;
            response = 1;
        }
        break;
        case 2:
        {
            toSendPackage->number =  signal2;
            response = 2;
        }
        break;
        default:
        {
            signal1 = receivedPackage->number;
            toSendPackage->number =  signal1;
            response = 1;
        }
    }

    return response;
}
```


DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

GUI Composer 2



DEPARTMENT OF ENERGY TECHNOLOGY
AALBORG UNIVERSITY

GUI Composer 2

- Online GUI Composer 2 Editor
  - https://dev.ti.com/gc/

- Getting Started
  - https://dev.ti.com/gc/designer/help/Tutorials/GettingStarted/index.html

- User guide
  - https://dev.ti.com/gc/designer/help/UsersGuide/index.html