DAG API and Programming Manual

Introduction

The APIs are written in ‘C’ language and the following manual assumes the knowledge of ‘C’ programming and build environment. The package will have driver, library and sample application using the API. The application will need to include ‘ctidaglib.h’ for all API declaration and will need to link static library libctidag.a, this library has the implementation of the entire API and communicates with the DAG board.

Connecting to DAG board

The application will need to connect the DAG board to use DAG board. API ‘ConnectDAGBoard’ will connect to the DAG board. The first argument is a pointer to ‘CtiUfpgaBoard’ is declared in ctidaglib.h, the second argument is board index to the system and is ‘0’ if no other DAG board is present in the system.

```
CtiUfpgaBoard brdInfo;
int ret;

ret = ConnectDAGBoard(&brdInfo, 0);
if(ret != CTI_STATUS_OK)
{
      printf("could not connect to DAG board error %x\n",ret);
}
```

Disconnecting from DAG board

brdInfo is obtained by a previous call to ConnectDAGBoard.

```
ret = DisconnectDAGBoard(&brdInfo)
if(ret != CTI_STATUS_OK)
{
      printf("could not disconnect from DAG board error %x\n",ret);
}
```

Configuring a ADC port

The application will use 'SetAdcCmd' API to configure an ADC port; the API has the following declaration

```
int SetAdcCmd(CtiUfpgaBoard brdInfo, unsigned char index, unsigned char
singleEnded, unsigned char channel, unsigned char polarity, unsigned char
inputRange);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
index - index to ADC port, index is from ADC0 to ADC3
singleEnded - Boolean value to indicate whether the input is single ended or
differential. 1 indicates single ended and 0 indicates differential
channel - channel of the input, the value range is from CHANNEL0 to CHANNEL7
polarity - Boolean value to indicate polarity of differential input. Please consult ADC
operation section of DAG manual for reference. This is the odd sign value and is only
used when the input is set to differential
inputRange - input range of ADC, the value range is from ADC_VP5N5 to ADC_VP10
```

The following code will set ADC0 port singled ended, input on CHANNEL0 and range 0 to+5V.

```
ret = SetAdcCmd(brdInfo, ADC0, 1, CHANNEL0, 0, ADC_VP5);
if(ret != CTI_STATUS_OK)
{
      printf("SetAdcCmd error %x\n",ret);
}
```

Reading ADC input

The application will use 'ReadAdc' API to read input from a ADC port; the API has the following declaration

```
int ReadAdc(CtiUfpgaBoard brdInfo, int index, float *value);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
index - index to ADC port, index is from ADC0 to ADC3
value – float pointer to the read value
```

The following code shows how ADC0 value is read in 'value'.

```
ret = ReadAdc(brdInfo, ADC0, &value);
if(ret != CTI_STATUS_OK)
{
      printf("ReadAdc error %x\n",ret);
}
```

Configuring a DAC port

The application will use 'SetConfigDac' API to configure a DAC port; the API has the following declaration

```
int SetConfigDac(CtiUfpgaBoard brdInfo, unsigned char index, unsigned char
range);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
index - index to DAC port, index is from DAC0 to DAC_ALL
range - output range of DAC, the value range is from DAC_VP5 to DAC_VP75N25
```

The following code will set DAC0 within range 0 to+5V.

```
ret = SetConfigDac(brdInfo, DAC0, DAC_VP5);
if(ret != CTI_STATUS_OK)
{
      printf("SetConfigDac error %x\n",ret);
}
```

Output DAC value

The application will use 'WriteDAC' API to output to a DAC port; the API has the following declaration

```
int WriteDAC(CtiUfpgaBoard brdInfo, unsigned char index, float value);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
index - index to DAC port, index is from DAC0 to DAC_ALL
value - output value
```

The following code will make DAC0 to output +4.5V.

```
ret = WriteDAC(brdInfo, DAC0, 4.5);
if(ret != CTI_STATUS_OK)
{
      printf("WriteDAC error %x\n",ret);
}
```

Configuring GPIO

The application will use 'SetGPIO' API to configure GPIO; the API has the following declaration

```
int SetGPIO(CtiUfpgaBoard brdInfo, unsigned short direction);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
direction - bit value of the direction for 16 GPIO. Output=0, Input=1.
```

The following code will set first 8 GPIO as outputs and the rest are inputs.

```
ret = SetGPIO(brdInfo, 0xff00);
if(ret != CTI_STATUS_OK)
{
	printf("SetGPIO error %x\n",ret);
}
```

Read GPIO input

The application will use '`InputGPIO`' API to read GPIO input; the API has the following declaration

```
int InputGPIO(CtiUfpgaBoard brdInfo, unsigned short *value);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
value - Pointer to bit value
```

The following code will get the GPIO input

```
ret = InputGPIO(brdInfo, &value);
if(ret != CTI_STATUS_OK)
{
	printf("InputGPIO error %x\n",ret);
}
```

Write GPIO output

The application will use '`OutputGPIO`' API to write GPIO output; the API has the following declaration

```
int OutputGPIO(CtiUfpgaBoard brdInfo, unsigned short value);
```

The arguments are as follows

```
brdInfo - CtiUfpgaBoard structure variable
value - bit value of the output
```

The following code will write the GPIO output

```
ret = OutputGPIO(brdInfo, value);
if(ret != CTI_STATUS_OK)
{
	printf("OutputGPIO error %x\n",ret);
}
```